Entwicklerhandbuch für SDK v2

AWS SDK for JavaScript

AWS SDK for JavaScript: Entwicklerhandbuch für SDK v2

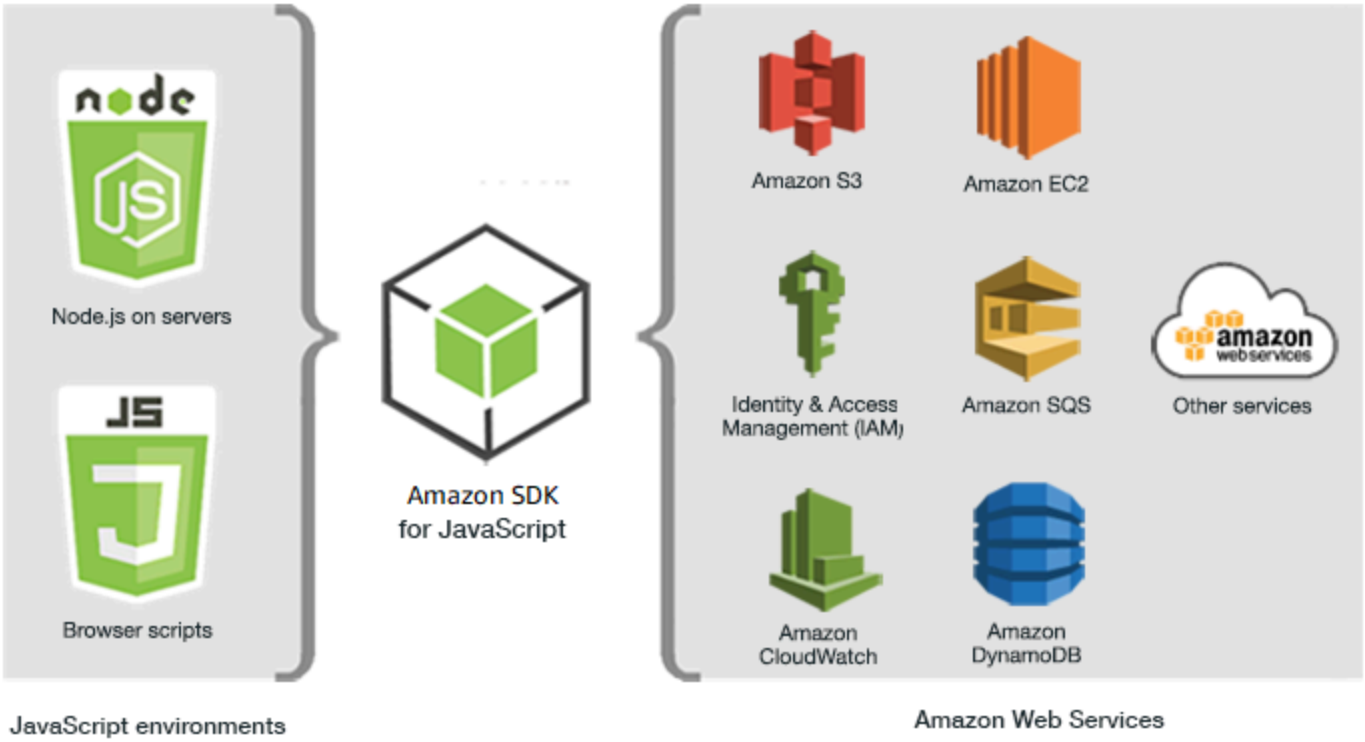
Table of Contents

Wir haben das Kommende end-of-support für AWS SDK for JavaScript v2 angekündigt. Wir empfehlen Ihnen, auf AWS SDK for JavaScript Version 3 zu migrieren. Termine, weitere Details und Informationen zur Migration finden Sie in der verlinkten Ankündigung.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.

Was ist das AWS SDK for JavaScript?

Das AWS SDK for JavaScriptstellt eine JavaScript API für AWS Dienste bereit. Sie können die JavaScript API verwenden, um Bibliotheken oder Anwendungen für Node.js oder den Browser zu erstellen.



Nicht alle Services sind sofort im SDK verfügbar. Informationen darüber, welche Dienste derzeit von der unterstützt werdenAWS SDK for JavaScript, finden Sie unter https://github.com/aws/ aws-sdk-js / blob/master/services.md. Informationen zum SDK für on finden Sie unter. JavaScript GitHub Weitere Ressourcen

Wartung und Support für SDK-Hauptversionen

Informationen zu Wartung und Support für SDK-Hauptversionen und deren zugrunde liegende Abhängigkeiten finden Sie im AWS-Referenzhandbuch zu SDKs und Tools:

- AWS-SDKs- und Tools-Wartung-Richtlinie
- Support-Matrix für AWS-SDKs- und Tools-Version

Verwenden des SDKs mit Node.js

Node.js ist eine plattformübergreifende Runtime für die Ausführung serverseitiger Anwendungen JavaScript . Sie können Node.js auf einer Amazon EC2 EC2-Instance für die Ausführung auf einem Server einrichten. Darüber hinaus können Sie mit Node.js On-Demand-AWS Lambda-Funktionen schreiben.

Die Verwendung des SDK für Node.js unterscheidet sich von der Art und Weise, wie Sie es JavaScript in einem Webbrowser verwenden. Der Unterschied hängt davon ab, wie das SDK geladen wird und wie die erforderlichen Anmeldeinformationen für den Zugriff auf bestimmte Web-Services abgerufen werden. Wenn sich die Verwendung bestimmter APIs zwischen Node.js und dem Browser unterscheidet, werden diese Unterschiede benannt.

Verwenden Sie das SDK mit AWS Cloud9

Sie können Node.js -Anwendungen auch mithilfe des SDK für JavaScript in der AWS Cloud9 IDE entwickeln. Ein Beispiel für die Verwendung AWS Cloud9 bei der Entwicklung von Node.js finden Sie unter Node.js Sample for AWS Cloud9 im AWS Cloud9Benutzerhandbuch. Weitere Informationen zur Verwendung AWS Cloud9 mit dem SDK für JavaScript finden Sie unterVerwenden von AWS Cloud9 mit der AWS SDK for JavaScript.

Verwenden des SDK mit AWS Amplify

Für browserbasierte Web-, Mobil- und Hybrid-Apps können Sie auch die AWSAmplify Library on verwenden GitHub, die das SDK für erweitert und eine JavaScript deklarative Schnittstelle bereitstellt.

> **Note**
>
> Frameworks wie AWS Amplify bieten möglicherweise nicht dieselbe Browserunterstützung wie das SDK für JavaScript. Weitere Informationen finden Sie in der Dokumentation eines Frameworks.

Verwenden des SDKs mit Webbrowsern

Alle gängigen Webbrowser unterstützen die Ausführung von. JavaScript JavaScriptCode, der in einem Webbrowser ausgeführt wird, wird oft als JavaScriptclientseitig bezeichnet.

Die Verwendung des SDK für JavaScript in einem Webbrowser unterscheidet sich von der Art und Weise, wie Sie es für Node.js verwenden. Der Unterschied hängt davon ab, wie das SDK geladen wird und wie die erforderlichen Anmeldeinformationen für den Zugriff auf bestimmte Web-Services abgerufen werden. Wenn sich die Verwendung bestimmter APIs zwischen Node.js und dem Browser unterscheidet, werden diese Unterschiede benannt.

Eine Liste der von AWS SDK for JavaScriptunterstützten Browser finden Sie unter Unterstützte Webbrowser.

Häufige Anwendungsfälle

Die Verwendung des SDK für JavaScript In-Browser-Skripte ermöglicht die Realisierung einer Reihe überzeugender Anwendungsfälle. Im Folgenden finden Sie einige Ideen für Dinge, die Sie in einer Browseranwendung erstellen können, indem Sie das SDK für den JavaScript Zugriff auf verschiedene Webdienste verwenden.

- Erstellen Sie eine benutzerdefinierte Konsole für AWS Dienste, in der Sie auf Funktionen aus verschiedenen Regionen und Diensten zugreifen und diese kombinieren können, um Ihre Organisations- oder Projektanforderungen bestmöglich zu erfüllen.
- Verwenden Sie Amazon Cognito Identity, um authentifizierten Benutzerzugriff auf Ihre Browseranwendungen und Websites zu ermöglichen, einschließlich der Verwendung der Drittanbieter-Authentifizierung von Facebook und anderen.
- Verwenden Sie Amazon Kinesis, um Klickstreams oder andere Marketingdaten in Echtzeit zu verarbeiten.
- Verwenden Sie Amazon DynamoDB für die serverlose Datenpersistenz, z. B. für individuelle Benutzereinstellungen für Website-Besucher oder Anwendungsbenutzer.
- Verwenden von AWS Lambda zum Einkapseln proprietärer Logik, die Sie über Browser-Skripts aufrufen können, ohne Ihr geistiges Eigentum herunterzuladen oder gegenüber Benutzern offenzulegen.

Informationen zu den Beispielen

Sie können das SDK in der JavaScript Codebeispielbibliothek nach Beispielen durchsuchen. AWS

Erste Schritte mit dem AWS SDK for JavaScript

Der AWS SDK for JavaScript ermöglicht den Zugriff auf Webdienste entweder in Browserskripten oder in Node.js. In diesem Abschnitt finden Sie zwei Übungen für den Einstieg, die Ihnen zeigen, wie Sie mit dem SDK für JavaScript jede dieser JavaScript Umgebungen arbeiten.

Sie können Node.js -Anwendungen auch mithilfe des SDK für JavaScript in der AWS Cloud9 IDE entwickeln. Ein Beispiel für die Verwendung AWS Cloud9 bei der Entwicklung von Node.js finden Sie unter Node.js Sample for AWS Cloud9 im AWS Cloud9 Benutzerhandbuch.

Themen

Erste Schritte in einem Browser-Skript

Dieses Beispiel eines Browser-Skripts zeigt Ihnen:

- So greifen Sie mit Amazon Cognito Identity über ein Browserskript auf AWS Dienste zu.
- So verwandeln Sie Text mit Amazon Polly in synthetisierte Sprache.
- Wie ein Presigner-Objekt verwendet wird, um eine vorsignierte URL zu erstellen.

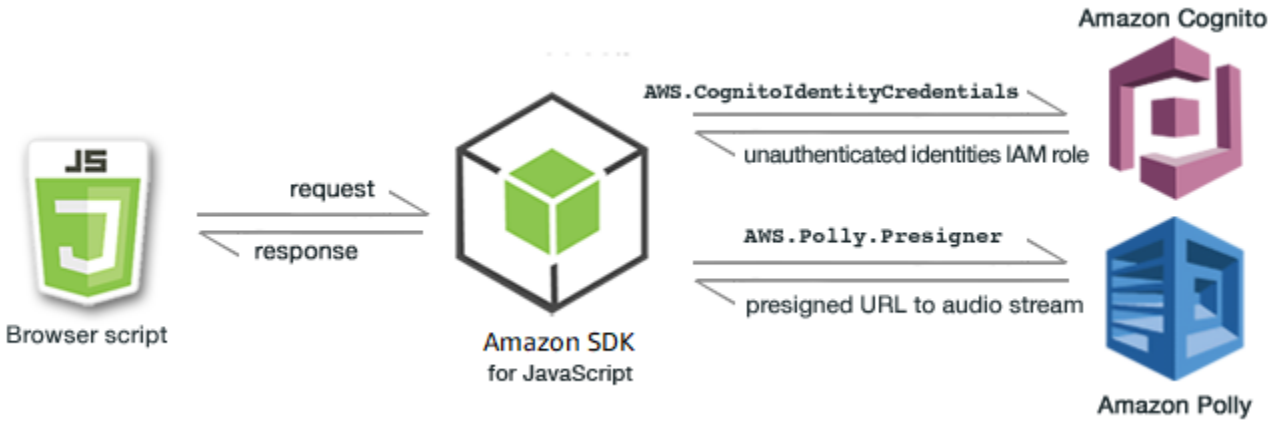
Das Szenario

Amazon Polly ist ein Cloud-Service, der Text in naturgetreue Sprache umwandelt. Sie können Amazon Polly verwenden, um Anwendungen zu entwickeln, die das Engagement und die Barrierefreiheit erhöhen. Amazon Polly unterstützt mehrere Sprachen und beinhaltet eine Vielzahl lebensechter Stimmen. Weitere Informationen zu Amazon Polly finden Sie im Amazon Polly Developer Guide.

Das Beispiel zeigt, wie Sie ein einfaches Browser-Skript einrichten und ausführen, das den von Ihnen eingegebenen Text an Amazon Polly sendet und dann die URL des synthetisierten Audios des

Textes zurückgibt, damit Sie ihn abspielen können. Das Browserskript verwendet Amazon Cognito Identity, um Anmeldeinformationen bereitzustellen, die für den Zugriff auf AWS Dienste erforderlich sind. Sie werden die grundlegenden Muster für das Laden und Verwenden des SDK JavaScript in Browserskripten sehen.

> **Note**
>
> Das Abspielen der Sprachausgabe in diesem Beispiel hängt davon ab, ob Sie einen Browser ausführen, der HTML 5-Audio unterstützt.



Das Browserskript verwendet das SDK, um Text mithilfe der JavaScript folgenden APIs zu synthetisieren:

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

Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool

In dieser Übung erstellen und verwenden Sie einen Amazon Cognito Cognito-Identitätspool, um nicht authentifizierten Zugriff auf Ihr Browserskript für den Amazon Polly bereitzustellen. Durch die Erstellung eines Identitätspools werden auch zwei IAM-Rollen erstellt, eine zur Unterstützung von Benutzern, die von einem Identitätsanbieter authentifiziert wurden, und die andere zur Unterstützung nicht authentifizierter Gastbenutzer.

In dieser Übung arbeiten wir nur mit der Rolle für nicht authentifizierte Benutzer, um uns auf die Aufgabe zu konzentrieren. Sie können die Unterstützung für einen Identitätsanbieter und authentifizierte Benutzer zu einem späteren Zeitpunkt integrieren. Weitere Informationen zum Hinzufügen eines Amazon Cognito-Identitätspools finden Sie unter Tutorial: Creating an Identity Pool im Amazon Cognito Developer Guide.

So erstellen Sie einen Amazon Cognito Cognito-Identitätspool

1. Melden Sie sich bei der Amazon Cognito Cognito-Konsole an AWS Management Console und öffnen Sie sie unter https://console.aws.amazon.com/cognito/.
2. Wählen Sie im linken Navigationsbereich Identity Pools aus.
3. Wählen Sie Identitätspool erstellen.
4. Wählen Sie unter Vertrauensstellung im Identitätspool konfigurieren die Option Gastzugriff für die Benutzerauthentifizierung aus.
5. Wählen Sie unter Berechtigungen konfigurieren die Option Neue IAM-Rolle erstellen aus und geben Sie einen Namen (z. B. getStartedRole) in den IAM-Rollennamen ein.
6. Geben Sie unter Eigenschaften konfigurieren einen Namen ein (z. B. getStartedPool) im Feld Identitätspoolname.
7. Bestätigen Sie unter Überprüfen und erstellen die Auswahl, die Sie für Ihren neuen Identitätspool getroffen haben. Wählen Sie Bearbeiten, um zum Assistenten zurückzukehren und Einstellungen zu ändern. Wählen Sie danach Identitätspool erstellen aus.
8. Notieren Sie sich die Identitätspool-ID und die Region des neu erstellten Amazon Cognito Cognito-Identitätspools. *Sie benötigen diese Werte, um IDENTITY_POOL_ID und REGION in zu ersetzen.* Schritt 4: Schreiben des Browser-Skripts

Nachdem Sie Ihren Amazon Cognito Cognito-Identitätspool erstellt haben, können Sie Berechtigungen für Amazon Polly hinzufügen, die von Ihrem Browserskript benötigt werden.

Schritt 2: Fügen Sie der erstellten IAM-Rolle eine Richtlinie hinzu

Um den Browserskriptzugriff auf Amazon Polly für die Sprachsynthese zu aktivieren, verwenden Sie die nicht authentifizierte IAM-Rolle, die für Ihren Amazon Cognito Cognito-Identitätspool erstellt wurde. Dazu müssen Sie der Rolle eine IAM-Richtlinie hinzufügen. Weitere Informationen zum Ändern von IAM-Rollen finden Sie unter Ändern einer Rollenberechtigungsrichtlinie im IAM-Benutzerhandbuch.

So fügen Sie der IAM-Rolle, die nicht authentifizierten Benutzern zugeordnet ist, eine Amazon Polly Polly-Richtlinie hinzu

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im linken Navigationsbereich Roles aus.
3. Wählen Sie den Namen der Rolle aus, die Sie ändern möchten (z. B. getStartedRole), und wählen Sie dann die Registerkarte Berechtigungen.
4. Wählen Sie Berechtigungen hinzufügen und anschließend Richtlinien anhängen aus.
5. Suchen Sie auf der Seite „Berechtigungen hinzufügen“ für diese Rolle nach dem Kontrollkästchen für und aktivieren Sie es AmazonPollyReadOnly.

   > **Note**
   >
   > Sie können diesen Prozess verwenden, um den Zugriff auf jeden AWS Dienst zu aktivieren.

6. Wählen Sie Add permissions (Berechtigungen hinzufügen) aus.

Nachdem Sie Ihren Amazon Cognito Cognito-Identitätspool erstellt und Ihrer IAM-Rolle für nicht authentifizierte Benutzer Berechtigungen für Amazon Polly hinzugefügt haben, können Sie die Webseite und das Browserskript erstellen.

Schritt 3: Erstellen der HTML-Seite

Die Beispielanwendung besteht aus einer einzelnen HTML-Seite, die die Benutzeroberfläche und das Browser-Skript enthält. Um zu beginnen, erstellen Sie ein HTML-Dokument und kopieren Sie in dieses den folgenden Inhalt. Die Seite enthält ein Eingabefeld und eine Schaltfläche, ein `<audio>`-Element zum Abspielen der Sprachausgabe und ein <p>-Element zum Anzeigen von Nachrichten. (Beachten Sie, dass das vollständige Beispiel unten auf dieser Seite angezeigt wird.)

Weitere Informationen zum `<audio>`-Element finden Sie unter 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>
```

Speichern Sie die HTML-Datei und nennen Sie diese `polly.html`. Nach dem Erstellen der Benutzeroberfläche für die Anwendung können Sie den Browser-Skriptcode hinzufügen, der die Anwendung ausführt.

Schritt 4: Schreiben des Browser-Skripts

Wenn Sie das Browser-Skript erstellen, müssen Sie zunächst das SDK für einbeziehen, JavaScript indem Sie ein `<script>` Element nach dem Element auf der `<audio>` Seite hinzufügen. Die aktuelle SDK_VERSION_NUMBER finden Sie in der API-Referenz für das SDK im API-Referenzhandbuch. JavaScript AWS SDK for JavaScript

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

Fügen Sie dann ein neues `<script type="text/javascript">`-Element nach dem SDK-Eintrag hinzu. Sie fügen das Browser-Skript diesem Element hinzu. Legen Sie die AWS Region und die Anmeldeinformationen für das SDK fest. Als Nächstes erstellen Sie eine Funktion mit dem Namen `speakText()`, die von der Schaltfläche als Ereignishandler aufgerufen wird.

Um Sprache mit Amazon Polly zu synthetisieren, müssen Sie eine Vielzahl von Parametern angeben, darunter das Soundformat der Ausgabe, die Abtastrate, die ID der zu verwendenden Stimme und den wiederzugeben Text. Wenn Sie erstmalig die Parameter erstellen, setzen Sie den `Text:`-Parameter auf eine leere Zeichenfolge; der `Text:`-Parameter wird auf den Wert gesetzt, den Sie aus dem `<input>`-Element auf der Webseite abgerufen haben. Ersetzen Sie *IDENTITY_POOL_ID*

und *REGION* im folgenden Code durch die unter angegebenen Werte. Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool

```
    <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 gibt synthetisierte Sprache als Audiostream zurück. Der einfachste Weg, dieses Audio in einem Browser abzuspielen, besteht darin, Amazon Polly das Audio unter einer vordefinierten URL zur Verfügung zu stellen, die Sie dann als `src` Attribut des `<audio>` Elements auf der Webseite festlegen können.

Erstellen Sie ein neues `AWS.Polly`-Serviceobjekt. Erstellen Sie dann das `AWS.Polly.Presigner`-Objekt, das Sie zum Erstellen der vorsignierten URL verwenden, von der die Sprachausgabe abgerufen werden kann. Sie müssen die Sprachparameter, die Sie definiert haben sowie das `AWS.Polly`-Serviceobjekt, das Sie erstellt haben, an den `AWS.Polly.Presigner`-Konstruktor übergeben.

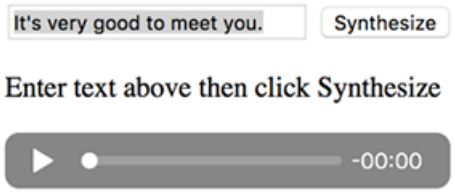
Nach dem Erstellen des Presigner-Objekts, rufen Sie die `getSynthesizeSpeechUrl`-Methode dieses Objekts auf und übergeben Sie die Sprachparameter. Wenn diese Aktion erfolgreich ist, gibt diese Methode die URL der Sprachausgabe zurück, die Sie dann dem `<audio>`-Element für die Wiedergabe zuweisen.

```
            // 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>
```

Schritt 5: Ausführen des Beispiels

Um die Beispielanwendung auszuführen, laden Sie `polly.html` in einen Webbrowser. So sollte die Browser-Präsentation ungefähr aussehen.



Geben Sie einen Satz in das Eingabefeld ein, der in Sprache umgewandelt werden soll, und wählen Sie dann Synthesize (Generieren) aus. Wenn das Audio zum Abspielen bereit ist, wird eine Meldung angezeigt. Verwenden Sie die Steuerelemente des Audio-Players zum Anhören der Sprachausgabe.

Vollständiges Beispiel

Hier finden Sie die vollständige HTML-Seite mit dem Browser-Skript. Es ist auch hier verfügbar. 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>
```

Mögliche Erweiterungen

Im Folgenden finden Sie Varianten dieser Anwendung, mit denen Sie die Verwendung des SDK JavaScript in einem Browserskript weiter untersuchen können.

- Experimentieren Sie mit anderen Sound-Ausgabeformaten.
- Fügen Sie die Option hinzu, eine der verschiedenen von Amazon Polly bereitgestellten Stimmen auszuwählen.
- Integrieren Sie einen Identitätsanbieter wie Facebook oder Amazon, um ihn mit der authentifizierten IAM-Rolle zu verwenden.

Erste Schritte in Node.js

Dieses Node.js-Codebeispiel zeigt:

- Wie das `package.json`-Manifest für Ihr Projekt erstellt wird.
- Wie die Module, die Ihr Projekt verwendet, installiert und eingeschlossen werden.
- So erstellen Sie ein Amazon Simple Storage Service (Amazon S3) -Serviceobjekt aus der `AWS.S3` Client-Klasse.
- So erstellen Sie einen Amazon S3 S3-Bucket und laden ein Objekt in diesen Bucket hoch.

Das Szenario

Das Beispiel zeigt, wie Sie ein einfaches Modul Node.js einrichten und ausführen, das einen Amazon S3 S3-Bucket erstellt und ihm dann ein Textobjekt hinzufügt.

Da Bucket-Namen in Amazon S3 global eindeutig sein müssen, beinhaltet dieses Beispiel ein Drittanbieter-Modul Node.js, das einen eindeutigen ID-Wert generiert, den Sie in den Bucket-Namen integrieren können. Dieses zusätzliche Modul trägt den Namen `uuid`.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Erstellen Sie ein Arbeitsverzeichnis zum Entwickeln Ihres Node.js-Moduls. Geben Sie diesem Verzeichnis den Namen `awsnodesample`. Beachten Sie, dass das Verzeichnis an einem Speicherort erstellt werden muss, der von Anwendungen aktualisiert werden kann. Beispiel: Erstellen Sie bei Windows das Verzeichnis nicht unter "C:\Programme".
- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website. Sie finden Downloads der aktuellen und LTS-Versionen von Node.js für eine Vielzahl von Betriebssystemen unter https://nodejs.org/en/download/current/.

Inhalt

Schritt 1: Installieren Sie das SDK und die Abhängigkeiten

Sie installieren das SDK für das JavaScript Paket mit npm (dem Paketmanager Node.js).

Geben Sie im `awsnodesample`-Verzeichnis im Paket den folgenden Befehl in die Befehlszeile ein.

```
npm install aws-sdk
```

Mit diesem Befehl wird das SDK für JavaScript in Ihrem Projekt installiert und aktualisiert`package.json`, sodass das SDK als Projektabhängigkeit aufgeführt wird. Sie finden Informationen zu diesem Paket , indem Sie nach "aws-sdk" auf der npm-Website suchen.

Als Nächstes installieren Sie das `uuid`-Modul in das Projekt, indem Sie den folgenden Befehl zur Installation des Moduls und Aktualisierung von `package.json` in die Befehlszeile eingeben. Weitere Informationen zu `uuid` finden Sie auf der Modulseite unter https://www.npmjs.com/package/uuid.

```
npm install uuid
```

Diese Pakete und der zugehörige Code werden im `node_modules`-Unterverzeichnis Ihres Projekts installiert.

Weitere Informationen zum Installieren von Node.js-Paketen finden Sie unter Herunterladen und Installieren von lokalen Paketen und Erstellen von Node.js-Modulen auf der Node.js-Paketmanager (npm)-Website. Informationen zum Herunterladen und Installieren von finden Sie unterInstallieren des SDK für JavaScript. AWS SDK for JavaScript

Schritt 2: Konfigurieren Sie Ihre Anmeldeinformationen

Sie müssen Anmeldeinformationen angeben, AWS damit das SDK nur auf Ihr Konto und die zugehörigen Ressourcen zugreifen kann. Weitere Informationen zum Abrufen Ihrer Kontoanmeldeinformationen finden Sie unter SDK-Authentifizierung mit AWS.

Um diese Informationen zu speichern, empfehlen wir Ihnen, eine gemeinsame Anmeldeinformationsdatei zu erstellen. Um zu erfahren wie dies geht, vgl. Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei. Ihre Anmeldeinformationsdatei sollte dem folgenden Beispiel ähneln.

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

Sie können feststellen, ob Sie Ihre Anmeldeinformationen korrekt festgelegt haben, indem Sie den folgenden Code mit Node.js ausführen:

```
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);
```

```
  }
});
```

Wenn Sie Ihre Region in Ihrer `config` Datei korrekt festgelegt haben, können Sie diesen Wert ebenfalls anzeigen, indem Sie die `AWS_SDK_LOAD_CONFIG` Umgebungsvariable auf einen beliebigen Wert setzen und den folgenden Code verwenden:

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

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

Schritt 3: Erstellen Sie das Paket-JSON für das Projekt

Nachdem Sie das `awsnodesample`-Projektverzeichnis erstellt haben, erstellen und fügen Sie eine `package.json`-Datei für die Speicherung der Metadaten für Ihr Node.js-Projekt hinzu. Einzelheiten zur Verwendung `package.json` in einem Node.js -Projekt finden Sie unter Erstellen einer package.json-Datei.

Erstellen Sie im Projektverzeichnis eine neue Datei namens `package.json`. Fügen Sie dann dieses JSON der Datei hinzu.

```
{
  "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"
}
```

Speichern Sie die Datei. Wenn Sie die benötigten Module installieren, wird der `dependencies`-Teil der Datei abgeschlossen. Eine JSON-Datei, die ein Beispiel für diese Abhängigkeiten zeigt, finden Sie hier unter. GitHub

Schritt 4: Schreiben des Node.js-Codes

Erstellen Sie eine neue Datei mit dem Namen `sample.js`, wohin Sie den Beispielcode speichern möchten. Fügen Sie zunächst die `require` Funktionsaufrufe hinzu, um das SDK für JavaScript und die `uuid` Module einzubeziehen, sodass sie für Sie verfügbar sind.

Erstellen Sie einen eindeutigen Bucket-Namen, der zur Erstellung eines Amazon S3 S3-Buckets verwendet wird, indem Sie in diesem Fall `'node-sdk-sample-'` einen eindeutigen ID-Wert an ein erkennbares Präfix anhängen. Sie generieren die eindeutige ID durch Aufrufen des `uuid`-Moduls. Erstellen Sie dann einen Namen für den `Key`-Parameter, der zum Hochladen eines Objekts in den Bucket verwendet wird.

Erstellen Sie ein `promise`-Objekt zum Aufrufen der `createBucket`-Methode des `AWS.S3`-Serviceobjekts. Bei einer erfolgreichen Antwort erstellen Sie die erforderlichen Parameter zum Hochladen von Text in den neu erstellten Bucket. Bei Verwendung eines anderen Promise rufen Sie die `putObject`-Methode auf, um das Textobjekt in den Bucket hochzuladen.

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

Diesen Beispielcode finden Sie hier auf GitHub.

Schritt 5: Ausführen des Beispiels

Geben Sie den folgenden Befehl ein, um das Beispiel auszuführen.

```
node sample.js
```

Wenn der Upload erfolgreich ist, sehen Sie eine Bestätigungsmeldung in der Befehlszeile. Sie können zudem den Bucket und das hochgeladene Textobjekt in der Amazon S3-Konsole finden.

Verwenden von AWS Cloud9 mit der AWS SDK for JavaScript

Sie können es verwendenAWS Cloud9mit demAWS SDK for JavaScriptSie können JavaScript im Browser-Code schreiben und ausführen — sowie Ihren Code von Node.js mit nur einem Browser schreiben, ausführen und debuggen.AWS Cloud9umfasst Tools wie einen Code-Editor und ein Terminal sowie einen Debugger für Node.js -Code. Da die AWS Cloud9 IDE Cloud-basiert ist, können Sie im Büro, zuhause oder unterwegs über das Internet an Ihren Projekten arbeiten. Allgemeine Informationen zu finden Sie unter.AWS Cloud9, finden Sie unterAWS Cloud9-Benutzerhandbuchaus.

Gehen Sie wie folgt vor, um einzurichtenAWS Cloud9mit dem SDK für JavaScript:

Inhalt

Schritt 1: Einrichten IhrerAWSZu verwendendes KontoAWS Cloud9

Beginnen Sie mit der NutzungAWS Cloud9indem Sie sich bei derAWS Cloud9-Konsole alsAWS Identity and Access Management(IAM) -Entity (z. B. ein IAM-Benutzer), die über Zugriffsberechtigungen für verfügtAWS Cloud9in IhremAWSKonto.

So richten Sie eine IAM-Entity in Ihrem einAWS-Konto zum ZugreifenAWS Cloud9, und um sich bei derAWS Cloud9-Konsole finden Sie unter.Team-Einrichtung fürAWS Cloud9imAWS Cloud9-Benutzerhandbuchaus.

Schritt 2: Einrichten IhrerAWS Cloud9Entwicklungsumgebung

Nachdem Sie sich bei der AWS Cloud9-Konsole angemeldet haben, verwenden Sie die Konsole, um eine AWS Cloud9-Entwicklungsumgebung zu erstellen. Nachdem Sie die Umgebung erstellt haben, öffnet AWS Cloud9 die IDE für diese Umgebung.

Siehe .Erstellen einer Umgebung inAWS Cloud9imAWS Cloud9-BenutzerhandbuchWeitere Informationen finden Sie unter.

> **Note**
>
> Wenn Sie Ihre Umgebung in der Konsole zum ersten Mal erstellen, empfehlen wir, dass Sie die Option zum Erstellen einer neuen Instance für die Umgebung (EC2) verwenden. Diese Option sagtAWS Cloud9Um eine Umgebung zu erstellen, starten Sie eine Amazon EC2 EC2-Instance und verbinden Sie dann die neue Instance mit der neuen Umgebung. Dies ist der schnellste Weg, mit der Arbeit mit AWS Cloud9 zu beginnen.

Schritt 3: Einrichten des -SDK für JavaScript

NachAWS Cloud9Öffnen Sie die IDE für Ihre Entwicklungsumgebung, befolgen Sie eine oder beide der folgenden Verfahren, um das SDK für JavaScript in Ihrer Umgebung einzurichten.

So richten Sie das SDK für JavaScript für Node.js ein

1. Wenn das Terminal in der IDE noch nicht geöffnet ist, öffnen Sie es. Wählen Sie in diesem Fall auf der Menüleiste in der IDE Window, New Terminal (Fenster, Neues Terminal) aus.
2. Führen Sie den folgenden Befehl aus, um npm zur Installation des -SDK für JavaScript zu verwenden.

   ```
   npm install aws-sdk
   ```

   Wenn die IDE npm nicht finden kann, führen Sie die folgenden Befehle nacheinander in der folgenden Reihenfolge aus, um npm zu installieren. (Diese Befehle setzen voraus, dass Sie die Option zum Erstellen einer neuen Instance für die Umgebung (EC2) ausgewählt haben, wie weiter oben in diesem Thema beschrieben.)

> **Warning**
>
> AWS hat keine Kontrolle über den folgenden Code. Bevor Sie ihn ausführen, überprüfen Sie unbedingt dessen Authentizität und Integrität. Weitere Informationen zu diesem Code finden Sie im GitHub-Repository 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).
```

So richten Sie das SDK für JavaScript im Browser ein

Sie müssen das SDK für JavaScript nicht installieren, um es in Browser-Skripts zu verwenden. Sie können das gehostete SDK for JavaScript Paket direkt von ladenAWSmit einem Skript in Ihren HTML-Seiten.

Sie können verkleinerte und nicht verkleinerte Versionen des aktuellen SDK für JavaScript von GitHub herunterladen unterhttps://github.com/aws/aws-sdk-js/tree/master/distaus.

Schritt 4: Herunterladen des Beispielcodes

Verwenden Sie das Terminal, das Sie im vorherigen Schritt geöffnet haben, um Beispielcode für das SDK für JavaScript in dasAWS Cloud9-Entwicklungsumgebung. (Wenn das Terminal nicht bereits in der IDE geöffnet ist, öffnen Sie es, indem Sie Window, New Terminal (Fenster, Neues Terminal) auf der Menüleiste in der IDE auswählen.)

Führen Sie den folgenden Befehl aus, um den Beispielcode herunterzuladen. Dieser Befehl lädt eine Kopie aller im offiziellen verwendeten Code-Beispiele herunterAWSSDK-Dokumentation in das Stammverzeichnis Ihrer Umgebung.

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

Um Codebeispiele für das SDK für JavaScript zu finden, verwenden Sie dieUmgebungFenster zum Öffnen des*ENVIRONMENT_NAME*`\aws-doc-sdk-examples\javascript\example_code`, wobei*UMWELTNAME*ist der Name IhresAWS Cloud9-Entwicklungsumgebung.

Weitere Informationen zum Arbeiten mit diesen und anderen Codebeispielen finden Sie unter.Beispiele für SDK for JavaScript Codeaus.

Schritt 5: Beispielcode ausführen und debuggen

So führen Sie Code in IhremAWS Cloud9Entwicklungsumgebung finden Sie unter.Ausführen Ihres CodesimAWS Cloud9-Benutzerhandbuchaus.

Informationen zum Debuggen von Node.js -Code finden Sie unterDebuggen Ihres CodesimAWS Cloud9-Benutzerhandbuchaus.

Einrichten des SDK für JavaScript

In den Themen in diesem Abschnitt wird erläutert, wie Sie das SDK für JavaScript zur Verwendung in Webbrowsern und mit Node.js installieren. Außerdem wird gezeigt, wie das SDK geladen wird, sodass Sie auf die vom SDK unterstützten Web-Services zugreifen können.

> **Note**
>
> React-Native-Entwickler sollten verwenden AWS Amplify , um neue Projekte auf zu erstellen AWS. Weitere Informationen finden Sie im aws-sdk-react-native Archiv.

Themen

Voraussetzungen

Bevor Sie die verwenden AWS SDK for JavaScript, stellen Sie fest, ob Ihr Code in Node.js oder Webbrowsern ausgeführt werden muss. Führen Sie anschließend die folgenden Schritte aus:

- Im Fall von Node.js installieren Sie Node.js auf Ihren Servern, wenn nicht bereits geschehen.
- Im Fall von Webbrowsern identifizieren Sie die Browserversionen, die Sie unterstützen müssen.

Themen

Einrichten einer AWS Node.js-Umgebung

Verwenden Sie eine der folgenden Methoden, um eine AWS Node.js-Umgebung einzurichten, in der Sie Ihre Anwendung ausführen können:

- Wählen Sie ein Amazon Machine Image (AMI) mit vorinstalliertem Node.js aus und erstellen Sie eine Amazon EC2-Instance mit diesem AMI. Wählen Sie beim Erstellen Ihrer Amazon EC2-Instance Ihr AMI aus der aus AWS Marketplace. Suchen Sie nach AWS Marketplace Node.js und wählen Sie eine AMI-Option aus, die eine vorinstallierte Version von Node.js (32-Bit oder 64-Bit) enthält.
- Erstellen Sie eine Amazon EC2-Instance und installieren Sie Node.js darauf. Weitere Informationen zum Installieren von Node.js auf einer Amazon Linux-Instance finden Sie unter Tutorial: Node.js auf einer Amazon EC2 EC2-Instance einrichten.
- Erstellen Sie eine Serverless-Umgebung mit AWS Lambda , um Node.js als Lambda-Funktion auszuführen. Weitere Informationen zur Verwendung von Node.js innerhalb einer Lambda-Funktion finden Sie unter Programmiermodell (Node.js) im AWS Lambda -Entwicklerhandbuch.
- Stellen Sie Ihre Node.js-Anwendung in bereit AWS Elastic Beanstalk. Weitere Informationen zur Verwendung von Node.js mit Elastic Beanstalk finden Sie unter Bereitstellen von Node.js-Anwendungen in AWS Elastic Beanstalk im AWS Elastic Beanstalk -Entwicklerhandbuch.
- Erstellen Sie einen Node.js-Anwendungsserver mit AWS OpsWorks. Weitere Informationen zur Verwendung von Node.js mit finden Sie AWS OpsWorks unter Erstellen Ihres ersten Node.js-Stacks im AWS OpsWorks -Benutzerhandbuch.

Unterstützte Webbrowser

Das SDK for JavaScript unterstützt alle modernen Webbrowser, einschließlich dieser Mindestversionen:

Browser	Version
Google Chrome	28.0+
Mozilla Firefox	26.0+
Oper	17.0+
Microsoft Edge	25.10+
Windows Internet Explorer	N/A
Apple Safari	5+

Browser	Version
Android-Browser	4.3+

> **Note**
>
> Frameworks wie AWS Amplify bieten möglicherweise nicht denselben Browser-Support wie das SDK für JavaScript. Weitere Informationen finden Sie in der Dokumentation eines Frameworks.

Installieren des SDK für JavaScript

Ob und wie Sie die installieren, AWS SDK for JavaScript hängt davon ab, ob der Code in Node.js-Modulen oder Browserskripten ausgeführt wird.

Nicht alle Services sind sofort im SDK verfügbar. Informationen dazu, welche Services derzeit von unterstützt werden AWS SDK for JavaScript, finden Sie unter https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md

Node

Die bevorzugte Methode zur Installation von AWS SDK for JavaScript für Node.js ist die Verwendung von npm, dem Node.js-Paketmanager . Geben Sie dazu Folgendes in der Befehlszeile ein.

```
npm install aws-sdk
```

Falls Sie folgende Fehlermeldung sehen:

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

Geben Sie die folgenden Befehle in der Befehlszeile ein:

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

Browser

Sie müssen das SDK zur Nutzung in Browser-Skripts nicht installieren. Sie können das gehostete SDK-Paket direkt von Amazon Web Services mit einem Skript auf Ihren HTML-Seiten laden. Das gehostete SDK-Paket unterstützt die Teilmenge der AWS Services, die Cross-Origin Resource Sharing (CORS) erzwingen. Weitere Informationen finden Sie unter Laden des SDK für JavaScript.

Sie können einen benutzerdefinierten Build des SDK erstellen, in dem Sie die spezifischen Web-Services und -Versionen, die Sie verwenden möchten, auswählen. Anschließend laden Sie Ihr benutzerdefiniertes SDK-Paket für die lokale Entwicklung herunter und hosten es für die Verwendung in Ihrer Anwendung. Weitere Informationen zum Erstellen eines benutzerdefinierten SDKs finden Sie unter Erstellen des SDK für Browser.

Sie können minifizierte und nichtminierte verteilbare Versionen des aktuellen AWS SDK for JavaScript von herunterladen GitHub unter:

https://github.com/aws/aws-sdk-js/Baum/Master/Verzeichnis

Installieren mit Bower

Bower ist ein Paket-Manager für das Web. Nachdem Sie Bower installiert haben, können Sie damit das SDK installieren. Geben Sie Folgendes in einem Terminalfenster ein, um das SDK mit Bower zu installieren:

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

Laden des SDK für JavaScript

Wie Sie das SDK für laden, JavaScript hängt davon ab, ob Sie es für die Ausführung in einem Webbrowser oder in Node.js laden.

Nicht alle Services sind sofort im SDK verfügbar. Informationen dazu, welche Services derzeit von unterstützt werden AWS SDK for JavaScript, finden Sie unter https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md

Node.js

Nachdem Sie das SDK installiert haben, können Sie das AWS Paket in Ihre Knotenanwendung mit ladenrequire.

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

React Native

Um das SDK in einem React Native Projekt zu verwenden, installieren Sie zunächst das SDK mit npm:

```
npm install aws-sdk
```

Verwenden Sie den folgenden Code, um in Ihrer Anwendung auf die mit React Native kompatible Version des SDKs zu verweisen:

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

Browser

Die schnellste Möglichkeit, mit dem SDK zu beginnen, besteht darin, das gehostete SDK-Paket direkt von Amazon Web Services zu laden. Um dies zu tun, fügen Sie ein <script> -Element in Ihre HTML-Seiten in der folgenden Form ein:

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

Informationen zum aktuellen SDK_VERSION_NUMBER finden Sie in der API-Referenz für das SDK für JavaScript im AWS SDK for JavaScript API-Referenzhandbuch für .

Nachdem das SDK in Ihrer Seite geladen wurde, ist es über die globale Variable AWS (oder window.AWS) verfügbar.

Wenn Sie zum Bündeln Ihrer Code- und Modulabhängigkeiten browserify verwenden, laden Sie das SDK mit require, genauso wie in Node.js.

Aktualisieren des SDK für JavaScript von Version 1

Die folgenden Hinweise helfen Ihnen beim Upgrade des SDK für JavaScript von Version 1 auf Version 2.

Automatische Konvertierung von Base64- und Zeitstempeltypen bei Eingabe/Ausgabe

Das SDK verschlüsselt und entschlüsselt nun Base64-kodierte Werte sowie Zeitstempelwerte automatisch für den Benutzer. Diese Änderung wirkt sich auf alle Operationen aus, in denen Base64- oder Zeitstempelwerte von einer Anforderung gesendet oder in einer Antwort zurückgegeben wurden, die Base64-kodierte Werte zulässt.

Benutzer-Code, der zuvor Base64 konvertierte, ist nicht mehr erforderlich. Als Base64 verschlüsselte Werte werden jetzt von Serverantworten als Buffer-Objekte zurückgegeben und können auch als Buffer-Eingabe übergeben werden. Beispiel: Die folgenden `SQS.sendMessage`-Parameter aus Version 1:

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

Können folgendermaßen neu geschrieben werden.

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

Hier sehen Sie, wie die Nachricht gelesen wird.

```
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.RequestId to response.requestId verschoben

Das SDK speichert nun Anforderungs-IDs für alle Services in einem konsistenten Ort im `response`-Objekt anstatt innerhalb der `response.data`-Eigenschaft. Dadurch wird die Konsistenz zwischen Services, die Anforderungs-IDs auf unterschiedliche Weise bereitstellen, verbessert. Auch die Umbenennung der `response.data.RequestId`-Eigenschaft in `response.requestId` (`this.requestId` innerhalb einer Callback-Funktion) ist eine wichtige Änderung.

Ändern Sie Ihren Code wie folgt:

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

In Folgendes:

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

Freigelegte Wrapper-Elemente

Wenn Sie `AWS.ElastiCache`, `AWS.RDS` oder `AWS.Redshift` verwenden, müssen Sie auf die Antwort über die Ausgabeeigenschaft der obersten Ebene in der Antwort für einige Operationen zugreifen.

Beispiel: Die `RDS.describeEngineDefaultParameters`-Methode gab Folgendes zurück.

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

Sie gibt nun Folgendes zurück.

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

Die Liste der betroffenen Operationen für jeden Service werden in der folgenden Tabelle aufgeführt.

Client-Klasse	Operationen

`AWS.ElastiCache`	`authorizeCacheSecurityGroupIngress`
	`createCacheCluster`
	`createCacheParameterGroup`
	`createCacheSecurityGroup`
	`createCacheSubnetGroup`
	`createReplicationGroup`
	`deleteCacheCluster`
	`deleteReplicationGroup`
	`describeEngineDefaultParameters`
	`modifyCacheCluster`
	`modifyCacheSubnetGroup`
	`modifyReplicationGroup`
	`purchaseReservedCacheNodesOffering`
	`rebootCacheCluster`
	`revokeCacheSecurityGroupIngress`

`AWS.RDS`	`addSourceIdentifierToSubscription`

Client-Klasse	Operationen
	authorizeDBSecurityGroupIngress copyDBSnapshot createDBInstance createDBInstanceReadReplica createDBParameterGroup createDBSecurityGroup createDBSnapshot createDBSubnetGroup createEventSubscription createOptionGroup deleteDBInstance deleteDBSnapshot deleteEventSubscription describeEngineDefaultParameters modifyDBInstance modifyDBSubnetGroup modifyEventSubscription modifyOptionGroup promoteReadReplica purchaseReservedDBInstances Offering rebootDBInstance

Client-Klasse	Operationen

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

Client-Klasse	Operationen
AWS.Redshift	authorizeClusterSecurityGroupIngress authorizeSnapshotAccess copyClusterSnapshot createCluster createClusterParameterGroup createClusterSecurityGroup createClusterSnapshot createClusterSubnetGroup createEventSubscription createHsmClientCertificate createHsmConfiguration deleteCluster deleteClusterSnapshot describeDefaultClusterParameters disableSnapshotCopy enableSnapshotCopy modifyCluster modifyClusterSubnetGroup modifyEventSubscription modifySnapshotCopyRetentionPeriod

Client-Klasse	Operationen

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

Gelöschte Client-Eigenschaften

Die Eigenschaften `.Client` und `.client` wurden aus Service-Objekten entfernt. Wenn Sie die `.Client`-Eigenschaft in einer Service-Klasse oder eine `.client`-Eigenschaft in einer Service-Objekt-Instance verwenden, entfernen Sie diese aus Ihrem Code.

Der folgende Code, der mit Version 1 des SDK für verwendet wird JavaScript:

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

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

sollte wie folgt geändert werden.

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

Konfiguration des SDK für JavaScript

Bevor Sie das SDK für JavaScript zum Aufrufen von Webdiensten mithilfe der API verwenden, müssen Sie das SDK konfigurieren. Sie müssen mindestens die folgenden Einstellungen konfigurieren:

- Die Region, in der Sie Dienste anfordern.
- Die Anmeldeinformationen, die Ihren Zugriff auf die SDK-Ressourcen autorisieren.

Zusätzlich zu diesen Einstellungen müssen Sie möglicherweise auch Berechtigungen für Ihre AWS Ressourcen konfigurieren. Sie können beispielsweise den Zugriff auf einen Amazon S3 S3-Bucket oder den Lesezugriff auf eine Amazon DynamoDB-Tabelle einschränken.

Das Referenzhandbuch für AWS SDKs und Tools enthält auch Einstellungen, Funktionen und andere grundlegende Konzepte, die vielen SDKs gemeinsam sind. AWS

In den Themen in diesem Abschnitt werden verschiedene Möglichkeiten beschrieben, das SDK JavaScript für Node.js und die JavaScript Ausführung in einem Webbrowser zu konfigurieren.

Themen

Verwenden des Global Configuration Object

Es gibt zwei Möglichkeiten, das SDK zu konfigurieren:

- Legen Sie die globale Konfiguration mithilfe von `AWS.Config` fest.

- Übergeben Sie zusätzliche Konfigurationsinformationen an ein Serviceobjekt.

Das Festlegen der globalen Konfiguration mithilfe von `AWS.Config` ist häufig zu Beginn einfacher, aber die Service-Level-Konfiguration bietet mehr Kontrolle über die einzelnen Services. Die über `AWS.Config` festgelegte globale Konfiguration bietet Standardeinstellungen, die auf die anschließend von Ihnen erstellten Serviceobjekte angewandt werden und deren Konfiguration vereinfachen. Sie können jedoch die Konfiguration der einzelnen Serviceobjekte aktualisieren, sollten Ihre Anforderungen von der globalen Konfiguration abweichen.

Einrichten der globalen Konfiguration

Nachdem Sie das `aws-sdk`-Paket in Ihren Code geladen haben, können Sie die globale Variable `AWS` verwenden, um auf die SDK-Klassen zuzugreifen und mit einzelnen Services zu interagieren. Das SDK enthält ein globales Konfigurationsobjekt, `AWS.Config`, über das Sie die SDK-Konfigurationseinstellungen angeben können, die von Ihrer Anwendung benötigt werden.

Konfigurieren Sie das SDK, indem Sie `AWS.Config`-Eigenschaften entsprechend Ihren Anwendungsanforderungen festlegen. Die folgende Tabelle bietet eine Übersicht über `AWS.Config`-Eigenschaften, die bei der Konfiguration des SDK häufig verwendet werden.

Konfigurationsoptionen	Beschreibung
`credentials`	Pflichtfeld Gibt die Anmeldeinformationen an, die für den Zugriff auf Services und Ressourcen verwendet werden.
`region`	Pflichtfeld Gibt die Region an, in der Anfragen für Dienste vorgenommen werden.
`maxRetries`	Optional. Gibt die maximale Anzahl der Wiederholungsversuche für eine bestimmte Anforderung an.
`logger`	Optional. Gibt ein Logger-Objekt an, auf das die Debugging-Informationen geschrieben wurden.
`update`	Optional. Aktualisiert die aktuelle Konfiguration mit neuen Werten.

Weitere Informationen zum Konfigurationsobjekt finden Sie Class: AWS.Configin der API-Referenz.

Globale Konfigurationsbeispiele

Sie müssen die Region und die Anmeldeinformationen in `AWS.Config` festlegen. Sie können diese Eigenschaften als Teil des `AWS.Config`-Konstruktors definieren, wie im folgenden Browser-Skriptbeispiel gezeigt:

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

Sie können diese Eigenschaften auch nach dem Erstellen von `AWS.Config` mithilfe der `update`-Methode festlegen, wie im folgenden Beispiel anhand der Aktualisierung der Region gezeigt:

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

Sie können Ihre Standard-Anmeldeinformationen abrufen, indem Sie die statische `getCredentials`-Methode `AWS.config` aufrufen:

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

Wenn Sie Ihre Region in Ihrer `config` Datei korrekt angegeben haben, erhalten Sie diesen Wert auf ähnliche Weise, indem Sie die `AWS_SDK_LOAD_CONFIG` Umgebungsvariable is auf einen beliebigen Wert setzen und die statische `region` Eigenschaft von aufrufen`AWS.config`:

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

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

Festlegen der Konfiguration nach Service

Auf jeden Dienst, den Sie im SDK verwenden, JavaScript wird über ein Dienstobjekt zugegriffen, das Teil der API für diesen Dienst ist. Um beispielsweise auf den Amazon S3 S3-Service zuzugreifen, erstellen Sie das Amazon S3 S3-Serviceobjekt. Sie können für einen bestimmten Service Konfigurationseinstellungen als Teil des Konstruktors für dieses Serviceobjekt definieren. Wenn Sie die Konfigurationswerte auf einem Serviceobjekt festlegen, übernimmt der Konstruktor alle Konfigurationswerte, die von `AWS.Config` verwendet werden, einschließlich der Anmeldeinformationen.

Wenn Sie beispielsweise auf Amazon EC2-Objekte in mehreren Regionen zugreifen müssen, erstellen Sie ein Amazon EC2-Serviceobjekt für jede Region und legen Sie dann die Regionskonfiguration für jedes Serviceobjekt entsprechend fest.

```
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'});
```

Sie können die Konfigurationswerte für einen bestimmten Service auch definieren, wenn Sie das SDK mithilfe von `AWS.Config` konfigurieren. Das globale Konfigurationsobjekt unterstützt viele servicespezifische Konfigurationsoptionen. Weitere Informationen zur servicespezifischen Konfiguration finden Sie `Class: AWS.Config` in der AWS SDK for JavaScript API-Referenz.

Unveränderliche Konfigurationsdaten

Globale Konfigurationsänderungen gelten für Anfragen für alle neu erstellten Serviceobjekte. Neu erstellte Serviceobjekte werden zunächst mit den aktuellen globalen Konfigurationsdaten und anschließend mit den lokalen Konfigurationsoptionen konfiguriert. Am globalen `AWS.config`-Objekt vorgenommene Aktualisierungen gelten nicht für zuvor erstellte Serviceobjekte.

Vorhandene Serviceobjekte müssen manuell mit neuen Konfigurationsdaten aktualisiert werden oder Sie müssen ein neues Serviceobjekt erstellen und verwenden, welches über die neuen Konfigurationsdaten verfügt. Das folgende Beispiel erstellt ein neues Amazon S3 S3-Serviceobjekt mit neuen Konfigurationsdaten:

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

Einstellung der AWS Region

Eine Region ist eine benannte Gruppe von AWS Ressourcen in demselben geografischen Gebiet. Ein Beispiel für eine Region ist `us-east-1` die Region USA Ost (Nord-Virginia). Sie geben eine Region an, wenn Sie das SDK JavaScript so konfigurieren, dass das SDK auf die Ressourcen in dieser Region zugreift. Einige -Services werden nur in bestimmten Regionen angeboten.

Das SDK für wählt standardmäßig JavaScript keine Region aus. Sie können jedoch die Region mithilfe einer Umgebungsvariablen, einer freigegebenen `config`-Datei oder dem globalen Konfigurationsobjekt definieren.

In einem Client-Klassenkonstruktor

Wenn Sie ein Dienstobjekt instanziieren, können Sie die Region für die Ressource als Teil des Client-Klassenkonstruktors angeben, wie hier dargestellt.

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

Verwenden des Global Configuration Object

Um die Region in Ihrem JavaScript Code festzulegen, aktualisieren Sie das `AWS.Config` globale Konfigurationsobjekt wie hier gezeigt.

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

Weitere Informationen zu aktuellen Regionen und verfügbaren Diensten in den einzelnen Regionen finden Sie unter AWS Regionen und Endpunkte in der Allgemeine AWS-Referenz.

Verwenden einer Umgebungsvariablen

Sie können die Region mithilfe der Umgebungsvariablen `AWS_REGION` festlegen. Wenn Sie diese Variable definieren, JavaScript liest das SDK für sie und verwendet sie.

Verwenden einer freigegebenen Konfigurationsdatei

Ähnlich wie Sie in der freigegebenen Anmeldeinformationsdatei Anmeldeinformationen für die Verwendung durch das SDK speichern können, haben Sie die Möglichkeit, die Region und andere

Konfigurationseinstellungen in einer freigegebenen Datei mit dem Namen `config` zu speichern, die von den SDKs verwendet wird. Wenn die `AWS_SDK_LOAD_CONFIG` Umgebungsvariable auf einen beliebigen Wert gesetzt wurde, sucht das SDK für beim Laden JavaScript automatisch nach einer `config` Datei. Wo Sie die `config`-Datei speichern, hängt von Ihrem Betriebssystem ab:

- Benutzer von Linux, macOS oder Unix: `~/.aws/config`
- Benutzer von Windows: `C:\Users\USER_NAME\.aws\config`

Wenn Sie noch keine freigegebene `config`-Datei haben, können Sie diese in dem angegebenen Verzeichnis erstellen. Im folgenden Beispiel werden sowohl die Region als auch das Ausgabeformat über die `config`-Datei definiert.

```
[default]
   region=us-east-1
   output=json
```

Weitere Informationen zur Verwendung gemeinsam genutzter Konfigurations- und Anmeldeinformationsdateien finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei oder Konfigurations- und Anmeldeinformationsdateien im AWS Command Line Interface Benutzerhandbuch.

Rangfolge zum Festlegen der Region

Die Rangfolge zum Festlegen der Region ist wie folgt:

- Wenn eine Region an einen Client-Klassenkonstruktor übergeben wird, wird diese Region verwendet. Ist dies nicht der Fall, dann...
- Wenn eine Region für das globale Konfigurationsobjekt festgelegt ist, wird diese Region verwendet. Ist dies nicht der Fall, dann...
- Wenn die Umgebungsvariable `AWS_REGION` ein wahrer Wert ist, wird diese Region verwendet. Ist dies nicht der Fall, dann...
- Wenn die Umgebungsvariable `AMAZON_REGION` ein wahrer Wert ist, wird diese Region verwendet. Ist dies nicht der Fall, dann...
- Wenn die `AWS_SDK_LOAD_CONFIG` Umgebungsvariable auf einen beliebigen Wert gesetzt ist und die Datei mit gemeinsamen Anmeldeinformationen (`~/.aws/credentials`oder der durch angegebene Pfad`AWS_SHARED_CREDENTIALS_FILE`) eine Region für das konfigurierte Profil enthält, wird diese Region verwendet. Ist dies nicht der Fall, dann...

- Wenn die `AWS_SDK_LOAD_CONFIG` Umgebungsvariable auf einen beliebigen Wert gesetzt ist und die Konfigurationsdatei (`~/.aws/config`oder der durch angegebene Pfad`AWS_CONFIG_FILE`) eine Region für das konfigurierte Profil enthält, wird diese Region verwendet.

Festlegen von benutzerdefinierten Endpunkten

Aufrufe von API-Methoden im SDK für JavaScript erfolgen an Service-Endpunkt-URIs. Standardmäßig werden diese Endpunkte in der Region erstellt, die Sie für Ihren Code konfiguriert haben. Es gibt jedoch Fälle, in denen Sie einen benutzerdefinierten Endpunkt für Ihre API-Aufrufe angeben müssen.

Endpunkt-Zeichenfolgeformat

Endpunktwerte sollten eine Zeichenfolge im folgenden Format sein:

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

Endpunkte für die Region ap-northeast-3

Die Region `ap-northeast-3` in Japan wird nicht in Aufzählungs-APIs für Regionen zurückgegeben, z. B. `EC2.describeRegions`. Um Endpunkte für diese Region zu definieren, befolgen Sie das zuvor beschriebene Format. Der Amazon EC2 EC2-Endpunkt für diese Region wäre also

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

Endpunkte für MediaConvert

Sie müssen einen benutzerdefinierten Endpunkt für die Verwendung mit MediaConvert erstellen. Jedes Kundenkonto ist einem eigenen Endpunkt zugewiesen, den Sie verwenden müssen. Hier ist ein Beispiel für die Verwendung eines benutzerdefinierten Endpunkts mit 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
```

```
};
```

Um den API-Endpunkt für Ihr Konto aufzurufen, lesen Sie den Abschnitt `MediaConvert.describeEndpoints` in der API-Referenz.

Stellen Sie sicher, dass die Region in Ihrem Code mit der Region in der benutzerdefinierten Endpunkt-URI übereinstimmt. Eine fehlende Übereinstimmung zwischen der Regionseinstellung und der benutzerdefinierten Endpunkt-URI könnte dazu führen, dass API-Aufrufe nicht ausgeführt werden können.

Weitere Informationen zu MediaConvert dieser `AWS.MediaConvert`Klasse finden Sie in der API-Referenz oder im AWS Elemental MediaConvert Benutzerhandbuch.

SDK-Authentifizierung mit AWS

Sie müssen festlegen, wie sich Ihr Code AWS bei der Entwicklung mit AWS -Services authentifiziert. Sie können den programmgesteuerten Zugriff auf AWS Ressourcen je nach Umgebung und verfügbarem AWS Zugriff auf unterschiedliche Weise konfigurieren.

Informationen zur Auswahl Ihrer Authentifizierungsmethode und deren Konfiguration für das SDK finden Sie unter Authentifizierung und Zugriff im Referenzhandbuch für AWS SDKs und Tools.

Wir empfehlen, dass neue Benutzer, die sich lokal weiterentwickeln und von ihrem Arbeitgeber keine Authentifizierungsmethode erhalten, diese einrichten AWS IAM Identity Center. Diese Methode beinhaltet die Installation von, AWS CLI um die Konfiguration zu vereinfachen und sich regelmäßig beim AWS Zugangsportal anzumelden. Wenn Sie sich für diese Methode entscheiden, sollte Ihre Umgebung die folgenden Elemente enthalten, nachdem Sie das Verfahren zur IAM Identity Center-Authentifizierung im Referenzhandbuch für AWS SDKs und Tools abgeschlossen haben:

- Die AWS CLI, mit der Sie eine AWS Access-Portal-Sitzung starten, bevor Sie Ihre Anwendung ausführen.
- Eine gemeinsam genutzte AWS`config` Datei mit einem `[default]` Profil mit einer Reihe von Konfigurationswerten, auf die vom SDK aus verwiesen werden kann. Den Speicherort dieser Datei finden Sie unter Speicherort der freigegebenen Dateien im Referenzhandbuch für AWS SDKs und Tools.
- Die gemeinsam genutzte `config` Datei legt die `region`Einstellung fest. Dies legt die Standardeinstellung AWS-Region fest, die das SDK für AWS Anfragen verwendet. Diese Region wird für SDK-Dienstanforderungen verwendet, für die keine zu verwendende Region angegeben ist.

- Das SDK verwendet die Konfiguration des SSO-Token-Anbieters des Profils, um Anmeldeinformationen abzurufen, bevor Anfragen an gesendet AWS werden. Der `sso_role_name` Wert, bei dem es sich um eine IAM-Rolle handelt, die mit einem IAM Identity Center-Berechtigungssatz verbunden ist, ermöglicht den Zugriff auf die in Ihrer AWS -Services Anwendung verwendeten.

  Die folgende `config` Beispieldatei zeigt ein Standardprofil, das mit der Konfiguration des SSO-Token-Anbieters eingerichtet wurde. Die `sso_session` Einstellung des Profils bezieht sich auf den benannten `sso-session`Abschnitt. Der `sso-session` Abschnitt enthält Einstellungen zum Initiieren einer AWS Access-Portal-Sitzung.

```
[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
```

Für das SDK für müssen Ihrer Anwendung JavaScript keine zusätzlichen Pakete (wie `SSO` und`SSOOIDC`) hinzugefügt werden, um die IAM Identity Center-Authentifizierung zu verwenden.

Starten Sie eine AWS Access-Portal-Sitzung

Bevor Sie eine Zugriffsanwendung ausführen AWS -Services, benötigen Sie eine aktive AWS Access-Portal-Sitzung, damit das SDK die IAM Identity Center-Authentifizierung zur Auflösung von Anmeldeinformationen verwenden kann. Abhängig von Ihrer konfigurierten Sitzungsdauer läuft Ihr Zugriff irgendwann ab und das SDK wird auf einen Authentifizierungsfehler stoßen. Um sich beim AWS Zugriffsportal anzumelden, führen Sie den folgenden Befehl in der aus AWS CLI.

```
aws sso login
```

Wenn Sie die Anweisungen befolgt haben und ein Standardprofil eingerichtet haben, müssen Sie den Befehl nicht mit einer `--profile` Option aufrufen. Wenn die Konfiguration Ihres SSO-Token-

Anbieters ein benanntes Profil verwendet, lautet der Befehl `aws sso login --profile named-profile`.

Führen Sie den folgenden AWS CLI Befehl aus, um optional zu testen, ob Sie bereits eine aktive Sitzung haben.

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

Wenn Ihre Sitzung aktiv ist, werden in der Antwort auf diesen Befehl das in der gemeinsam genutzten `config` Datei konfigurierte IAM Identity Center-Konto und der Berechtigungssatz gemeldet.

> **Note**
>
> Wenn Sie bereits eine aktive AWS Access-Portal-Sitzung haben und ausführen`aws sso login`, müssen Sie keine Anmeldeinformationen angeben.
> Beim Anmeldevorgang werden Sie möglicherweise aufgefordert, den AWS CLI Zugriff auf Ihre Daten zu gewähren. Da AWS CLI das auf dem SDK für Python aufbaut, können Berechtigungsnachrichten Variationen des `botocore` Namens enthalten.

Weitere Authentifizierungsinformationen

Menschliche Benutzer, auch bekannt als menschliche Identitäten, sind die Personen, Administratoren, Entwickler, Betreiber und Verbraucher Ihrer Anwendungen. Sie müssen über eine Identität verfügen, um auf Ihre AWS Umgebungen und Anwendungen zugreifen zu können. Menschliche Benutzer, die Mitglieder Ihres Unternehmens sind, also Sie, der Entwickler, werden als Personalidentitäten bezeichnet.

Verwenden Sie beim Zugriff AWS temporäre Anmeldeinformationen. Sie können einen Identitätsanbieter für Ihre menschlichen Benutzer verwenden, um Verbundzugriff auf AWS Konten zu ermöglichen, indem Sie Rollen übernehmen, die temporäre Anmeldeinformationen bereitstellen. Für eine zentralisierte Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center (IAM Identity Center) zu verwenden, um den Zugriff auf Ihre Konten und die Berechtigungen innerhalb dieser Konten zu verwalten. Weitere Alternativen finden Sie im Folgenden:

- Weitere Informationen zu bewährten Methoden finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.

- Informationen zum Erstellen kurzfristiger AWS Anmeldeinformationen finden Sie unter Temporäre Sicherheitsanmeldeinformationen im IAM-Benutzerhandbuch.
- Weitere Informationen zu anderen SDKs für Anbieter von JavaScript Anmeldeinformationen finden Sie unter Standardisierte Anbieter von Anmeldeinformationen im Referenzhandbuch für AWS SDKs und Tools.

Festlegen von Anmeldeinformationen

AWS verwendet Anmeldeinformationen, um festzustellen, wer Dienste anruft und ob der Zugriff auf die angeforderten Ressourcen zulässig ist.

Unabhängig davon, ob Ihr JavaScript Code in einem Webbrowser oder auf einem Node.js -Server ausgeführt wird, muss er gültige Anmeldeinformationen erhalten, bevor er über die API auf Dienste zugreifen kann. Anmeldeinformationen können für das Konfigurationsobjekt entweder mithilfe von `AWS.Config` global oder pro Service festgelegt werden, indem Sie die Anmeldeinformationen direkt an ein Serviceobjekt übergeben.

Es gibt mehrere Möglichkeiten, Anmeldeinformationen festzulegen, die sich zwischen Node.js und JavaScript in Webbrowsern unterscheiden. Die Themen in diesem Abschnitt beschreiben, wie Sie die Anmeldeinformationen in Node.js oder Web-Browsern definieren. In jedem Fall werden die Optionen in der empfohlenen Reihenfolge dargestellt.

Bewährte Methoden für Anmeldeinformationen

Durch das ordnungsgemäße Festlegen von Anmeldeinformationen wird sichergestellt, dass Ihre Anwendung oder Ihr Browser-Skript auf die benötigten Services und Ressourcen zugreifen kann. Gleichzeitig werden Sicherheitsrisiken minimiert, die geschäftskritische Anwendungen beeinträchtigen oder vertrauliche Daten kompromittieren könnten.

Ein wichtiger Grundsatz beim Einrichten von Anmeldeinformationen lautet, immer nur die geringstmöglichen Berechtigungen zu erteilen, die für eine Aufgabe erforderlich sind. Es ist sicherer, minimale Berechtigungen für Ihre Ressourcen bereitzustellen und bei Bedarf weitere Berechtigungen hinzuzufügen, anstatt Berechtigungen bereitzustellen, die die geringsten Rechte überschreiten, und aufgrund dessen zu einem späteren Zeitpunkt möglicherweise Sicherheitsprobleme beheben zu müssen. Wenn Sie beispielsweise nicht einzelne Ressourcen lesen und schreiben müssen, z. B. Objekte in einem Amazon S3 S3-Bucket oder einer DynamoDB-Tabelle, legen Sie für diese Berechtigungen nur Lesen fest.

Weitere Informationen zur Gewährung der geringsten Rechte finden Sie im Abschnitt „Geringste Rechte gewähren“ des Themas „Bewährte Methoden“ im IAM-Benutzerhandbuch.

> **Warning**
>
> Obwohl es möglich ist, eine Hartcodierung Ihrer Anmeldeinformationen in einer Anwendung oder einem Browser-Skript vorzunehmen, wird hiervon abgeraten. Durch die harte Codierung von Anmeldeinformationen besteht die Gefahr, dass vertrauliche Informationen preisgegeben werden.

Weitere Informationen zur Verwaltung Ihrer Zugriffsschlüssel finden Sie unter Bewährte Methoden für die Verwaltung von AWS Zugriffsschlüsseln in der Allgemeine AWS-Referenz.

Themen

Einrichten der Anmeldeinformationen in Node.js

Es gibt mehrere Möglichkeiten in Node.js, um dem SDK die Anmeldeinformationen bereitzustellen. Einige davon sind sicherer und andere bieten eine höhere Benutzerfreundlichkeit bei der Anwendungsentwicklung. Beim Abrufen von Anmeldeinformationen in Node.js sollten Sie vorsichtig sein, wenn Sie sich auf mehr als eine Quelle verlassen, wie z. B. auf eine von Ihnen geladene Umgebungsvariable und eine JSON-Datei. Sie können die Berechtigungen, unter denen Ihr Code ausgeführt wird, ändern, ohne sich dieser Änderung bewusst zu sein.

So können Sie Ihre Anmeldeinformationen angeben (in der Reihenfolge der Stärke der Empfehlung):

1. Aus AWS Identity and Access Management (IAM-) Rollen für Amazon EC2 geladen
2. Aus der freigegebenen Anmeldeinformationsdatei (`~/.aws/credentials`) geladen
3. Aus den Umgebungsvariablen geladen
4. Aus einer JSON-Datei auf einem Datenträger geladen
5. Andere vom SDK bereitgestellte Klassen von Anmeldeinformationsanbietern JavaScript

Wenn mehr als eine Quelle für Anmeldeinformationen für die SDK verfügbar ist, lautet die Standard-Auswahlpriorität wie folgt:

1. Anmeldeinformationen, die explizit über den Service-Client-Konstruktor festgelegt werden
2. Umgebungsvariablen
3. Die freigegebene Anmeldeinformationsdatei
4. Anmeldeinformationen, die aus dem ECS-Anmeldeinformationsanbieter geladen werden (wenn zutreffend)
5. Anmeldeinformationen, die mithilfe eines Anmeldeinformationsprozesses abgerufen werden, der in der gemeinsamen AWS Konfigurationsdatei oder der Datei mit den gemeinsamen Anmeldeinformationen angegeben ist. Weitere Informationen finden Sie unter the section called “Anmeldeinformationen, die über einen Prozess für die Konfigurierung von Anmeldeinformationen geladen werden”.
6. Anmeldeinformationen, die mithilfe des Anmeldeinformationsanbieters der Amazon EC2 EC2-Instance aus AWS IAM geladen wurden (sofern in den Instance-Metadaten konfiguriert)

Weitere Informationen finden Sie unter `Class: AWS.Credentials`und `Class: AWS.CredentialProviderChain`in der API-Referenz.

> **Warning**
>
> Dies ist zwar möglich, wir empfehlen jedoch nicht, Ihre AWS Anmeldeinformationen in Ihrer Anwendung fest zu codieren. Die Hartcodierung der Anmeldeinformationen setzt die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel dem Risiko einer Offenlegung aus.

Die Themen in diesem Abschnitt beschreiben, wie Sie die Anmeldeinformationen in Node.js laden.

Themen

- Anmeldeinformationen in Node.js aus IAM-Rollen für Amazon EC2 laden
- Laden der Anmeldeinformationen für eine Node.js-Lambda-Funktion
- Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei
- Laden der Anmeldeinformationen in Node.js aus den Umgebungsvariablen
- Laden der Anmeldeinformationen in Node.js aus einer JSON-Datei

- Laden von Anmeldeinformationen in Node.js über einen Prozess für konfigurierte Anmeldeinformationen

Anmeldeinformationen in Node.js aus IAM-Rollen für Amazon EC2 laden

Wenn Sie Ihre Anwendung Node.js auf einer Amazon EC2-Instance ausführen, können Sie IAM-Rollen für Amazon EC2 nutzen, um automatisch Anmeldeinformationen für die Instance bereitzustellen. Wenn Sie Ihre Instance für die Verwendung von IAM-Rollen konfigurieren, wählt das SDK automatisch die IAM-Anmeldeinformationen für Ihre Anwendung aus, sodass Sie keine Anmeldeinformationen manuell angeben müssen.

Weitere Informationen zum Hinzufügen von IAM-Rollen zu einer Amazon EC2 EC2-Instance finden Sie unter Verwenden von IAM-Rollen für Amazon EC2 EC2-Instances im Referenzhandbuch für AWS SDKs und Tools.

Laden der Anmeldeinformationen für eine Node.js-Lambda-Funktion

Wenn Sie eine AWS Lambda Funktion erstellen, müssen Sie eine spezielle IAM-Rolle erstellen, die über die Berechtigung zum Ausführen der Funktion verfügt. Diese Rolle wird Ausführungsrolle genannt. Wenn Sie eine Lambda-Funktion einrichten, müssen Sie die von Ihnen erstellte IAM-Rolle als entsprechende Ausführungsrolle angeben.

Die Ausführungsrolle stellt der Lambda-Funktion die Anmeldeinformationen zur Verfügung, die sie zum Ausführen und Aufrufen anderer Webdienste benötigt. Daher müssen Sie keine Anmeldeinformationen für den Code Node.js angeben, den Sie in einer Lambda-Funktion schreiben.

Weitere Informationen zum Erstellen einer Lambda-Ausführungsrolle finden Sie unter Berechtigungen verwalten: Verwenden einer IAM-Rolle (Ausführungsrolle) im AWS Lambda Entwicklerhandbuch.

Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei

Sie können Ihre AWS Anmeldedaten in einer gemeinsam genutzten Datei speichern, die von SDKs und der Befehlszeilenschnittstelle verwendet wird. Wenn das SDK für JavaScript geladen wird, durchsucht es automatisch die Datei mit den gemeinsamen Anmeldeinformationen, die den Namen „Anmeldeinformationen“ trägt. Wo Sie die freigegebene Anmeldeinformationsdatei speichern, hängt von Ihrem Betriebssystem ab:

- Die Datei mit gemeinsam genutzten Anmeldeinformationen auf Linux, Unix und macOS: `~/.aws/credentials`

- Die freigegebene Datei für Anmeldeinformationen auf Windows: `C:\Users\USER_NAME\.aws\credentials`

Wenn Sie noch keine freigegebene Anmeldeinformationsdatei haben, finden Sie weitere Informationen unter SDK-Authentifizierung mit AWS. Sobald Sie die folgenden Anweisungen ausgeführt haben, sollten Sie Text ähnlich dem folgenden in der Anmeldeinformationsdatei sehen, wobei *<YOUR_ACCESS_KEY_ID>* Ihre Zugriffsschlüssel-ID und *<YOUR_SECRET_ACCESS_KEY>* Ihr geheimer Zugriffsschlüssel ist:

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

Ein Beispiel für die Verwendung dieser Datei finden Sie unter Erste Schritte in Node.js.

Unter der `[default]`-Abschnittsüberschrift wird ein Standardprofil sowie die zugehörigen Werte für die Anmeldeinformationen festgelegt. Sie können zusätzliche Profile in der gleichen freigegebenen Konfigurationsdatei erstellen, wobei diese über ihre eigenen Anmeldeinformationen verfügen. Das folgende Beispiel zeigt eine Konfigurationsdatei mit dem Standardprofil und zwei zusätzlichen Profilen:

```
[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>
```

Standardmäßig prüft das SDK die Umgebungsvariable `AWS_PROFILE`, um zu bestimmen, welches Profil verwendet werden soll. Wenn die `AWS_PROFILE`-Variable in Ihrer Umgebung nicht festgelegt ist, verwendet das SDK die Anmeldeinformationen für das `[default]`-Profil. Um eines der alternativen Profile zu verwenden, müssen Sie den Wert der Umgebungsvariablen `AWS_PROFILE` einstellen oder ändern. Beispiel: Um die Anmeldeinformationen aus dem Arbeitskonto bei der oben

gezeigten Konfigurationsdatei zu verwenden, legen Sie die `AWS_PROFILE` Umgebungsvariable auf `work-account` (passend für Ihr Betriebssystem).

> **Note**
>
> Wenn Sie Umgebungsvariablen einstellen, stellen Sie sicher, dass Sie später die angemessenen Schritte durchführen (gemäß den Anforderungen Ihres Betriebssystems), um die Variablen in der Befehlszeile oder der Befehlsumgebung verfügbar zu machen.

Nachdem Sie die Umgebungsvariable festgelegt haben (falls erforderlich), können Sie eine JavaScript Datei ausführen, die das SDK verwendet, z. B. eine Datei mit dem Namen`script.js`.

```
$ node script.js
```

Sie können auch das vom SDK verwendete Profil explizit auswählen. Definieren Sie dazu entweder `process.env.AWS_PROFILE` vor dem Laden der SDK oder wählen Sie den Anmeldeinformationsanbieter aus, wie im folgenden Beispiel gezeigt:

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

Laden der Anmeldeinformationen in Node.js aus den Umgebungsvariablen

Das SDK erkennt automatisch AWS Anmeldeinformationen, die in Ihrer Umgebung als Variablen festgelegt sind, und verwendet sie für SDK-Anfragen, sodass Sie keine Anmeldeinformationen in Ihrer Anwendung verwalten müssen. Die Umgebungsvariablen, die Sie zum Bereitstellen Ihrer Anmeldeinformationen festlegen, sind:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_SESSION_TOKEN`

Weitere Informationen zur Einstellung von Umgebungsvariablen finden Sie unter Unterstützung von Umgebungsvariablen im Referenzhandbuch für AWS SDKs und Tools.

Laden der Anmeldeinformationen in Node.js aus einer JSON-Datei

Sie können die Konfigurations- und Anmeldeinformationen aus einem JSON-Dokument auf der Festplatte mithilfe von `AWS.config.loadFromPath` laden. Der angegebene Pfad ist relativ zum aktuellen Arbeitsverzeichnis Ihres Vorgangs. Um zum Beispiel Anmeldeinformationen aus einer `'config.json'`-Datei mit folgendem Inhalt zu laden:

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

Verwenden Sie dann den folgenden Code:

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

> **Note**
>
> Durch das Laden der Konfigurationsdaten aus einem JSON-Dokument werden alle vorhandenen Konfigurationsdaten zurückgesetzt. Fügen Sie nach Verwendung dieser Methode zusätzliche Konfigurationsdaten hinzu. Das Laden von Anmeldeinformationen aus einem JSON-Dokument wird in Browser-Skripts nicht unterstützt.

Laden von Anmeldeinformationen in Node.js über einen Prozess für konfigurierte Anmeldeinformationen

Sie können Anmeldeinformationen unter Verwendung einer Methode beschaffen, die nicht in das SDK integriert ist. Geben Sie dazu in der gemeinsam genutzten Konfigurationsdatei oder der Datei mit den AWS gemeinsamen Anmeldeinformationen einen Prozess für Anmeldeinformationen an. Wenn die `AWS_SDK_LOAD_CONFIG` Umgebungsvariable auf einen beliebigen Wert gesetzt ist, bevorzugt das SDK den in der Konfigurationsdatei angegebenen Prozess gegenüber dem in der Anmeldeinformationsdatei angegebenen Prozess (falls vorhanden).

Einzelheiten zur Angabe eines Prozesses mit Anmeldeinformationen in der gemeinsamen AWS Konfigurationsdatei oder der Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie in der AWS CLI Befehlsreferenz, insbesondere in den Informationen zur Beschaffung von Anmeldeinformationen aus externen Prozessen.

Informationen zur Verwendung der `AWS_SDK_LOAD_CONFIG`-Umgebungsvariablen finden Sie unter the section called “Verwenden einer freigegebenen Konfigurationsdatei” in diesem Dokument.

Festlegen von Anmeldeinformationen in einem Web-Browser

Es gibt mehrere Möglichkeiten, um dem SDK die Anmeldeinformationen aus Browser-Skripts bereitzustellen. Einige davon sind sicherer und andere bieten eine höhere Benutzerfreundlichkeit bei der Entwicklung eines Skripts. So können Sie Ihre Anmeldeinformationen angeben (in der Reihenfolge der Stärke der Empfehlung):

1. Authentifizieren von Benutzern mithilfe von Amazon Cognito Identity und Bereitstellen der Anmeldeinformationen
2. Verwenden von Web-Verbundidentitäten
3. Hartcodierung im Skript

> **Warning**
>
> Es ist nicht empfehlenswert,AWSAnmeldeinformationen in Ihren Skripten. Die Hartcodierung der Anmeldeinformationen setzt die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel dem Risiko einer Offenlegung aus.

Themen

Authentifizieren von Benutzern mithilfe von Amazon Cognito Identity

Der empfohlene Weg zu erhaltenAWSAnmeldeinformationen für Ihre Browserskripts sind die Verwendung des Amazon Cognito Identity Credentials - Objekt`AWS.CognitoIdentityCredentials`aus. Amazon Cognito ermöglicht die Authentifizierung von Benutzern über Identitätsanbieter von Drittanbietern.

Um Amazon Cognito Identity verwenden zu können, müssen Sie zuerst einen Identitätspool in der Amazon Cognito Cognito-Konsole erstellen. Ein Identitäten-Pool stellt die Gruppe von Identitäten dar, die Ihre Anwendung für Ihre Benutzer bereitstellt. Die Identitäten, die den Benutzern zugewiesen

wurden, identifizieren eindeutig die einzelnen Benutzerkonten. Amazon-Cognito-Identitäten sind keine Anmeldeinformationen. Sie werden über die Unterstützung des Web-Identitätsverbunds in AWS Security Token Service (AWS STS) gegen Anmeldeinformationen ausgetauscht.

Amazon Cognito unterstützt Sie dabei, die Abstraktion von Identitäten über mehrere Identitätsanbieter hinweg mithilfe des`AWS.CognitoIdentityCredentials`-Objekt. Die geladene Identität wird dann gegen Anmeldeinformationen in AWS STS ausgetauscht.

Konfigurieren des Amazon Cognito Cognito-Identitäts-Objekts

Falls Sie es bisher noch nicht getan haben, erstellen Sie einen Identitäten-Pool für die Verwendung mit Ihren Browser-Skripts imAmazon-Cognito-Konsolebevor Sie konfigurieren`AWS.CognitoIdentityCredentials`aus. Erstellen und verknüpfen Sie sowohl authentifizierte als auch nicht authentifizierte IAM-Rollen für Ihren Identitäten-Pool.

Für nicht authentifizierte Benutzer wird die Identität nicht verifiziert, sodass diese Rolle für Gastbenutzer Ihrer Anwendung geeignet ist, oder in Fällen, in denen es keine Rolle spielt, ob Benutzer ihre Identität verifizieren lassen. Authentifizierte Benutzer melden sich bei Ihrer Anwendung über einen Drittanbieter an, der ihre Identität überprüft. Vergewissern Sie sich, dass Sie die Berechtigungen der Ressourcen entsprechend anpassen, damit Sie keinen Zugriff von nicht authentifizierten Benutzern darauf gewähren.

Nachdem Sie einen Identitäten-Pool mit angefügten Identitätsanbietern konfiguriert haben, können Sie mit `AWS.CognitoIdentityCredentials` Benutzer authentifizieren. Um die Anmeldeinformationen für Ihre Anwendung so zu konfigurieren, dass Sie `AWS.CognitoIdentityCredentials` verwenden können, setzen Sie die `credentials`-Eigenschaft für `AWS.Config` oder Sie verwenden eine servicespezifische Konfiguration. Im folgenden Beispiel wird verwende `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'
  }
});
```

Die optionale `Logins`-Eigenschaft ist eine Abbildung der Namen des Identitätsanbieters auf die Identitäts-Token für diese Anbieter. Wie Sie den Token von Ihrem Identitätsanbieter erhalten,

hängt davon ab, welchen Anbieter Sie verwenden. Ist beispielsweise Facebook einer Ihrer Identitätsanbieter, könnten sie die `FB.login`-Funktion aus dem Facebook SDK verwenden, um ein Identitätsanbieter-Token zu erhalten:

```
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.');
  }
});
```

Wechseln von nicht authentifizierten Benutzern zu authentifizierten Benutzern

Amazon Cognito unterstützt sowohl authentifizierte als auch nicht authentifizierte Benutzer. Nicht authentifizierte Benutzer erhalten Zugriff auf Ihre Ressourcen, auch wenn sie nicht über Ihre Identitätsanbieter angemeldet sind. Dieser Grad des Zugriffs ist nützlich, um Inhalte für Benutzer anzuzeigen, bevor diese sich anmelden. Jeder nicht authentifizierte Benutzer hat eine eindeutige Identität in Amazon Cognito, auch wenn er nicht individuell angemeldet und authentifiziert ist.

Anfänglich nicht authentifizierter Benutzer

Benutzer beginnen in der Regel mit der nicht authentifizierten Rolle, für die Sie die Eigenschaft für die Anmeldeinformationen Ihres Konfigurationsobjekts ohne eine `Logins`-Eigenschaft festlegen. In diesem Fall könnte Ihre Standardkonfiguration folgendermaßen aussehen:

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

Wechseln Sie zum authentifizierten Benutzer

Wenn ein nicht authentifizierter Benutzer sich bei einem Identitätsanbieter anmeldet und Sie ein Token haben, können Sie den Benutzer von einem nicht authentifizierten in einen authentifizierten Benutzer umändern, indem Sie eine benutzerdefinierte Funktion aufrufen, welche das Anmeldeobjekt aktualisiert und das `Logins`-Token hinzufügt:

```
// 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;
}
```

Darüber hinaus können Sie ein `CognitoIdentityCredentials`-Objekt erstellen. Wenn Sie dies tun, müssen Sie die Eigenschaften für die Anmeldeinformationen der vorhandenen, von Ihnen erstellten Serviceobjekte zurücksetzen. Serviceobjekte lesen die globale Konfiguration nur während der Objektinitialisierung.

Weitere Informationen zu`CognitoIdentityCredentials`Objekt, siehe`AWS.CognitoIdentityCredentials`imAWS SDK for JavaScriptAPI-Referenz.

Verwenden von Web-Verbundidentitäten zum Authentifizieren von Benutzern

Sie können einzelne Identitätsanbieter direkt für den Zugriff konfigurierenAWS-Ressourcen mithilfe eines Web-IdentitätsverbAWSunterstützt derzeit das Authentifizieren von Benutzern mithilfe eines Web-Identitätsverbunds über mehrere Identitätsanbieter:

- Login mit Amazon
- Facebook-Anmeldung
- Google-Anmeldung

Sie müssen Ihre Anwendung zunächst mit den Anbietern registrieren, die von Ihrer Anwendung unterstützt werden. Erstellen Sie anschließend eine IAM-Rolle und richten Sie die Berechtigungen dafür ein. Die IAM-Rolle, die Sie erstellen, wird dann für die Erteilung der Berechtigungen verwendet, die Sie für sie über den jeweiligen Identitätsanbieter konfiguriert haben. Sie können beispielsweise

eine Rolle so einrichten, dass Benutzer, die sich über Facebook anmelden, Lesezugriff auf einen bestimmten Amazon S3 S3-Bucket haben, den Sie steuern.

Nachdem Sie sowohl über eine IAM-Rolle mit konfigurierten Berechtigungen verfügen als auch eine Anwendung bei den von Ihnen ausgewählten Identitätsanbieter registriert haben, können Sie das SDK so einrichten, dass die Anmeldeinformationen für die IAM-Rolle mithilfe von Hilfscode folgendermaßen abgerufen werden:

```
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
});
```

Der Wert des `ProviderId`-Parameters hängt vom angegebenen Identitätsanbieter ab. Der Wert des `WebIdentityToken`-Parameters ist das abgerufene Zugriffstoken aus einer erfolgreichen Anmeldung mit dem Identitätsanbieter. Weitere Informationen zum Konfigurieren und Abrufen von Zugriffstoken für die einzelnen Identitätsanbieter finden Sie in der Dokumentation des Identitätsanbieters.

Schritt 1: Registrierung bei Identitätsanbietern

Registrieren Sie eine Anwendung zu Beginn bei den Identitätsanbietern, die Sie unterstützen möchten. Sie werden um Informationen gebeten, anhand derer Ihre Anwendung und möglicherweise auch deren Autor identifiziert werden können. Auf diese Weise wird sichergestellt, dass die Identitätsanbieter den Empfänger ihrer Benutzerinformationen kennen. In jedem Fall erstellt der Identitätsanbieter eine Anwendungs-ID, mit der Sie Benutzerrollen konfigurieren können.

Schritt 2: Erstellen einer IAM-Rolle für einen Identitätsanbieter

Nachdem Sie die Anwendungs-ID von einem Identitätsanbieter erhalten haben, rufen Sie die IAM-Konsole unterhttps://console.aws.amazon.com/iam/um eine neue IAM-Rolle zu erstellen.

So erstellen Sie eine IAM-Rolle für einen Identitätsanbieter

1. Rufen Sie in der Konsole den Abschnitt Roles (Rollen) auf und wählen Sie anschließend Create new role (Neue Rolle erstellen) aus.
2. Geben Sie einen Namen für die neue Rolle ein, über den Sie deren Verwendung verfolgen können, z. B. **`facebookIdentity`**. Wählen Sie anschließend Next Step (Nächster Schritt) aus.

3. Wählen Sie unter Select Role Type (Rollentyp auswählen) die Option Role for Identity Provider Access (Rolle für den Zugriff von Identitätsanbietern) aus.
4. Wählen Sie unter Grant access to web identity providers (Web-Identitätsanbietern Zugriff gewähren) die Option Select (Auswählen) aus.
5. AusIdentitätsanbieterWählen Sie den Identitätsanbieter aus, den Sie für diese IAM-Rolle verwenden möchten.

6. Geben Sie die Anwendungs-ID des Identitätsanbieters im Feld Application ID (Anwendungs-ID) ein und wählen Sie dann Next Step (Nächster Schritt) aus
7. Konfigurieren Sie Berechtigungen für die Ressourcen, die Sie bereitstellen möchten, damit der Zugriff auf bestimmte Operationen für bestimmte Ressourcen gewährt wird. Weitere Informationen zu IAM-Berechtigungen finden Sie unterÜbersicht überAWSIAM-BerechtigungenimIAM User Guideaus. Überprüfen Sie die Vertrauensstellung der Rolle und passen Sie sie erforderlichenfalls an. Klicken Sie anschließend auf Next Step (Nächster Schritt).
8. Fügen Sie zusätzliche benötigte Richtlinien an und wählen Sie dann Next Step (Nächster Schritt) aus. Weitere Informationen zu IAM-Richtlinien finden Sie unterÜbersicht über IAM-RichtlinienimIAM User Guideaus.
9. Prüfen Sie die neue Rolle und wählen Sie dann Create Role (Rolle erstellen) aus.

Sie können der Rolle andere Einschränkungen hinzufügen, z. B. die Zuordnung zu bestimmten Benutzer-IDs. Wenn die Rolle Schreibberechtigungen für Ihre Ressourcen erteilt, stellen Sie sicher, dass Sie die Rolle nur Benutzern mit den entsprechenden Berechtigungen zuordnen, ansonsten kann jeder Benutzer mit einer Amazon-, Facebook- oder Google-Identität Änderungen an Ihren Ressourcen vornehmen.

Weitere Informationen zur Verwendung des Web-Identitätsverbunds in IAM finden Sie unterÜber Web-IdentitätsverbundimIAM User Guideaus.

Schritt 3: Abrufen eines Zugriffstokens für Anbieter nach der Anmeldung

Richten Sie die Anmeldung für Ihre Anwendung ein, indem Sie das SDK des Identitätsanbieters verwenden. Sie können ein JavaScript-SDK des Identitätsanbieters herunterladen und installieren, das Ihnen die Benutzeranmeldung mithilfe von OAuth oder OpenID ermöglicht. Weitere Informationen zum Herunterladen und Einrichten des SDK-Codes in Ihrer Anwendung finden Sie in der SDK-Dokumentation Ihres Identitätsanbieters:

- Login mit Amazon
- Facebook-Anmeldung
- Google-Anmeldung

Schritt 4: Abrufen temporärer Anmeldeinformationen

Nachdem Sie die Anwendung, Rollen und Ressourcenberechtigungen konfiguriert haben, fügen Sie Ihrer Anwendung den Code hinzu, um temporäre Anmeldeinformationen zu erhalten. Diese Anmeldeinformationen werden unter Verwendung des Web-Identitätsverbunds über AWS Security Token Service bereitgestellt. Benutzer melden sich beim Identitätsanbieter an, wodurch Ihnen ein Zugriffstoken bereitgestellt wird. Einrichten der`AWS.WebIdentityCredentials`-Objekt mithilfe des ARN für die IAM-Rolle, die Sie für diesen Identitätsanbieter erstellt haben:

```
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
});
```

Serviceobjekte, die anschließend erstellt werden, verfügen über die entsprechenden Anmeldeinformationen. Objekte, die vor der Einrichtung der `AWS.config.credentials`-Eigenschaft erstellt wurden, verfügen nicht über die aktuellen Anmeldeinformationen.

Sie können `AWS.WebIdentityCredentials` auch vor dem Abrufen des Zugriffstokens erstellen. Auf diese Weise können Sie Serviceobjekte erstellen, die von den Anmeldeinformationen abhängen, bevor Sie das Zugriffstoken laden. Erstellen Sie hierzu das Anmeldeinformationsobjekt ohne den `WebIdentityToken`-Parameter:

```
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;
```

Definieren Sie anschließend `WebIdentityToken` im Callback des SDK des Identitätsanbieters, das den Zugriffstoken enthält:

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

Beispiele von Web-Verbundidentitäten

Im Folgenden finden Sie einige Beispiele zur Verwendung von Web-Verbundidentitäten, um Anmeldeinformationen im JavaScript des Browsers abzurufen. Diese Beispiele müssen über ein Host-Schema „http://“ oder „https://“ ausgeführt werden, um sicherzustellen, dass die Weiterleitung an Ihre Anwendung durch den Identitätsanbieter erfolgen kann.

Beispiel für Login with Amazon

Der folgende Code zeigt, wie Sie Login with Amazon als Identitätsanbieter verwenden.

```
<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>
```

Beispiel für eine Facebook-Anmeldung

Der folgende Code zeigt, wie Sie die Facebook-Anmeldung als Identitätsanbieter verwenden:

```
<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>
```

Beispiel für eine Google+-Anmeldung

Der folgende Code zeigt, wie Sie die Google+-Anmeldung als Identitätsanbieter verwenden. Das Zugriffstoken für den Web-Identitätsverbund von Google wird im `response.id_token` und nicht im `access_token` gespeichert, wie dies bei anderen Identitätsanbietern der Fall ist.

```
<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>
```

Schützen der API-Versionen

AWS Dienste haben API-Versionsnummern, um die API-Kompatibilität im Auge zu behalten. API-Versionen in AWS Diensten werden durch eine `YYYY-mm-dd` formatierte Datumszeichenfolge identifiziert. Die aktuelle API-Version für Amazon S3 ist beispielsweise`2006-03-01`.

Es wird empfohlen, die API-Version für einen Service zu schützen, wenn diese für den Produktions-Code erforderlich ist. Dies kann Ihre Anwendungen von Service-Änderungen isolieren, die aufgrund von Aktualisierungen des SDK erfolgen. Wenn Sie beim Erstellen von Serviceobjekten keine API-Version angeben, verwendet das SDK standardmäßig die neueste API-Version. Dies kann dazu führen, dass Ihre Anwendung eine aktualisierte API referenziert, die Änderungen enthält, die sich negativ auf Ihre Anwendung auswirken.

Um die API-Version, die Sie für einen Service verwenden, zu schützen, übergeben Sie den `apiVersion`-Parameter bei der Erstellung des Serviceobjekts. Im folgenden Beispiel wird ein neu erstelltes `AWS.DynamoDB`-Serviceobjekt der API-Version `2011-12-05` zugeordnet:

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

Sie können auch eine Reihe von Service-API-Versionen global konfigurieren, indem Sie den `apiVersions`-Parameter in `AWS.Config` angeben. Um beispielsweise bestimmte Versionen der DynamoDB- und Amazon EC2 EC2-APIs zusammen mit der aktuellen Amazon Redshift Redshift-API festzulegen, legen Sie Folgendes fest: `apiVersions`

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

Abrufen von API-Versionen

Informationen zum Abrufen der API-Version für einen Service finden Sie im Abschnitt Sperren der API-Version auf der Referenzseite des Services, z. B. https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html für Amazon S3.

Überlegungen zu Node.js

Der Code Node.js ist es zwar JavaScript, aber die Verwendung von AWS SDK for JavaScript in Node.js kann sich von der Verwendung des SDK in Browserskripts unterscheiden. Einige API-Methoden funktionieren in Node.js, jedoch nicht in Browser-Skripts und umgekehrt. Die erfolgreiche Verwendung einiger APIs hängt davon ab, wie vertraut Sie mit häufigen Node.js-Codierungsmustern sind, beispielsweise dem Importieren und Verwenden anderer Node.js-Module wie des `File System (fs)`-Moduls.

Verwenden integrierter Node.js-Module

Node.js bietet eine Reihe von integrierten Modulen, die Sie verwenden können ohne sie installieren zu müssen. Um diese Module verwenden zu können, müssen Sie ein Objekt mit der `require`-Methode erstellen und den Modulnamen angeben. Wenn beispielsweise das integrierte HTTP-Modul enthalten sein soll, geben Sie Folgendes ein.

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

Rufen Sie Methoden des Moduls ab, als würde es sich um Methoden dieses Objekts handeln. Das folgende Beispiel zeigt Code, der eine HTML-Datei liest.

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

Eine vollständige Liste aller integrierten Module, die Node.js bereitstellt, finden Sie in der Dokumentation für Node.js v6.11.1 auf der Node.js-Website.

Verwenden von NPM-Paketen

Zusätzlich zu den integrierten Modulen können Sie auch Drittanbieter-Code von npm, dem Node.js-Paketmanager, integrieren. Hierbei handelt es sich um ein Repository mit Open-Source-Node.js-Paketen sowie um eine Befehlszeilen-Schnittstelle für die Installation dieser Pakete. Weitere Informationen über npm und eine Liste der aktuell verfügbaren Pakete finden Sie unter https://www.npmjs.com. Weitere Informationen zu weiteren Node.js -Paketen, die Sie verwenden können, finden Sie hier unter GitHub.

Ein Beispiel für ein npm-Paket, das Sie mit dem verwenden können, AWS SDK for JavaScript ist`browserify`. Details hierzu finden Sie unter Erstellen des SDK als Abhängigkeit mit Browserify. Ein weiteres Beispiel ist `webpack`. Details hierzu finden Sie unter Bündeln von Anwendungen mit Webpack.

Themen

- Konfigurieren von maxSockets in Node.js
- Wiederverwenden von Verbindungen mit Keep-Alive in Node.js
- Konfigurieren von Proxys für Node.js
- Registrieren von Zertifikat-Bundles in Node.js

Konfigurieren von maxSockets in Node.js

In Node.js können Sie die maximale Anzahl der Verbindungen pro Ursprungsserver festlegen. Wenn `maxSockets` festgelegt wurde, setzt der Low-Level-HTTP-Client die Anforderungen auf eine Warteliste und ordnet sie Sockets zu, sobald diese verfügbar werden.

Mit dieser Option können Sie eine Obergrenze für die Anzahl der gleichzeitigen Anforderungen an einen bestimmten Ursprungsserver angeben. Indem Sie diesen Wert senken, kann die Anzahl der maximalen Ablehnungen oder Timeout-Fehler reduziert werden. Es kann jedoch auch die Speichernutzung erhöhen, da Anforderungen so lange in eine Warteschlange gesetzt werden, bis ein Socket verfügbar wird.

Das folgende Beispiel zeigt, wie Sie `maxSockets` für alle Serviceobjekte definieren, die Sie erstellen. In diesem Beispiel sind bis zu 25 gleichzeitiger Verbindungen zu jedem Service-Endpunkt erlaubt.

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

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

Das Gleiche kann auch für die einzelnen Services vorgenommen werden.

```
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
   }
});
```

Wenn Sie den Standardwert von `https` verwenden, übernimmt das SDK den `maxSockets`-Wert aus `globalAgent`. Wenn der `maxSockets`-Wert nicht definiert wurde oder `Infinity` ist, nimmt das SDK einen `maxSockets`-Wert von 50 an.

Weitere Informationen zum Festlegen von `maxSockets` in Node.js finden Sie in der Online-Dokumentation für Node.js.

Wiederverwenden von Verbindungen mit Keep-Alive in Node.js

Standardmäßig erstellt der standardmäßige Node.js-HTTP/HTTPS-Agent eine neue TCP-Verbindung für jede neue Anforderung. Um die Kosten für den Aufbau einer neuen Verbindung zu vermeiden, können Sie eine vorhandene Verbindung wiederverwenden.

Bei kurzlebigen Vorgängen, z. B. DynamoDB-Abfragen, ist der Latenzaufwand beim Einrichten einer TCP-Verbindung möglicherweise höher als der Vorgang selbst. Da DynamoDB-Verschlüsselung im Ruhezustand in AWS KMS integriert ist, kann es außerdem zu Latenzen kommen, weil die Datenbank für jeden Vorgang neue AWS KMS Cacheeinträge einrichten muss.

Die einfachste Methode, das SDK für die Wiederverwendung von TCP-Verbindungen JavaScript zu konfigurieren, besteht darin, die `AWS_NODEJS_CONNECTION_REUSE_ENABLED` Umgebungsvariable auf zu setzen. 1 Diese Funktion wurde in der Version 2.463.0 hinzugefügt.

Alternativ können Sie die `keepAlive`-Eigenschaft eines HTTP- oder HTTPS-Agenten auf `true` festlegen, wie im folgenden Beispiel gezeigt.

```
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
  }
});
```

Das folgende Beispiel zeigt, wie nur `keepAlive` für einen DynamoDB-Client festgelegt wird:

```
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
  }
});
```

Wenn `keepAlive` aktiviert ist, können Sie auch die anfängliche Verzögerung für TCP-Keep-Alive-Pakete mit `keepAliveMsecs` festlegen, die standardmäßig 1000 ms beträgt. Weitere Informationen finden Sie in der Node.js-Dokumentation .

Konfigurieren von Proxys für Node.js

Wenn Sie keine direkte Verbindung zum Internet herstellen können, JavaScript unterstützt das SDK für die Verwendung von HTTP- oder HTTPS-Proxys über einen HTTP-Agenten eines Drittanbieters, z. B. einen Proxy-Agent. Um den Proxy-Agenten zu installieren, geben Sie Folgendes in die Befehlszeile ein.

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

Wenn Sie einen anderen Proxy verwenden möchten, befolgen Sie die Anweisungen zur Installation und Konfiguration für diesen Proxy. Um diesen oder einen anderen Drittanbieter-Proxy in Ihrer Anwendung verwenden zu können, müssen Sie die `httpOptions`-Eigenschaft von `AWS.Config` so einrichten, dass der von Ihnen gewählte Proxy angegeben ist. Dieses Beispiel veranschaulicht `proxy-agent`.

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

Weitere Informationen zu anderen Proxy-Bibliotheken finden Sie im Abschnitt zu npm, dem Paketmanager von Node.js.

Registrieren von Zertifikat-Bundles in Node.js

Die Standard-Vertrauensspeicher für Node.js enthalten die Zertifikate, die für den Zugriff auf Dienste erforderlich sind. AWS In einigen Fällen könnte es vorteilhaft sein, nur eine bestimmte Gruppe von Zertifikaten einzuschließen.

In diesem Beispiel wird ein bestimmtes Zertifikat auf der Festplatte verwendet, mit dem ein `https.Agent` erstellt wird, der alle Verbindungen ablehnt, die nicht über das vorgesehene Zertifikat verfügen. Der neu erstellte `https.Agent` wird anschließend zur Aktualisierung der SDK-Konfiguration verwendet.

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

Überlegungen zum Browser-Skript

In den folgenden Themen werden besondere Überlegungen zur Verwendung von AWS SDK for JavaScript In-Browser-Skripts beschrieben.

Themen

- Erstellen des SDK für Browser
- Cross-Origin Resource Sharing (CORS)

Erstellen des SDK für Browser

Das SDK für JavaScript wird als JavaScript Datei mit Unterstützung für eine Reihe von Standarddiensten bereitgestellt. Diese Datei wird in der Regel mithilfe eines `<script>`-Tags in

Browser-Skripts geladen, das das gehostete SDK-Paket referenziert. Möglicherweise benötigen Sie Unterstützung für andere Services als die im Standardsatz enthaltenen Services oder müssen das SDK auf andere Weise anpassen.

Wenn Sie mit dem SDK außerhalb einer Umgebung arbeiten, in der CORS in Ihrem Browser durchgesetzt wird, und wenn Sie Zugriff auf alle vom SDK bereitgestellten Dienste für haben möchten JavaScript, können Sie lokal eine benutzerdefinierte Kopie des SDK erstellen, indem Sie das Repository klonen und dieselben Build-Tools ausführen, mit denen die gehostete Standardversion des SDK erstellt wird. In den folgenden Abschnitten werden die Schritte erklärt, anhand derer das SDK zusammen mit zusätzlichen Services und API-Versionen erstellt werden kann.

Themen

Verwenden Sie den SDK Builder zum Erstellen des SDK für JavaScript

Der einfachste Weg, Ihren eigenen Build von zu erstellen, AWS SDK for JavaScript ist die Verwendung der SDK Builder-Webanwendung unter https://sdk.amazonaws.com/builder/js. Verwenden Sie den SDK-Builder, um die Services und deren API-Versionen, die in Ihrem Build eingeschlossen werden sollen, anzugeben.

Wählen Sie zu Beginn Select all services (Alle Services auswählen) oder Select Standard Services (Standard-Services auswählen) aus. Hier haben Sie die Möglichkeit, Services hinzuzufügen oder zu entfernen. Wählen Sie Development (Entwicklung) aus, wenn Sie einen besser lesbaren Code wünschen, oder Minified (Minimiert), um einen minimierten Build bereitzustellen. Nachdem Sie die gewünschten Services und Versionen ausgewählt haben, wählen Sie Build aus, um Ihr benutzerdefiniertes SDK zu erstellen und herunterzuladen.

Verwenden der CLI zum Erstellen des SDK für JavaScript

Um das SDK für die JavaScript Verwendung von zu erstellen AWS CLI, müssen Sie zuerst das Git-Repository klonen, das die SDK-Quelle enthält. Git und Node.js müssen auf Ihrem Computer installiert sein.

Klonen Sie zunächst das Repository aus dem Verzeichnis GitHub und ändern Sie das Verzeichnis in das Verzeichnis:

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

Nachdem Sie das Repository geklont haben, laden Sie das Abhängigkeitsmodul für das SDK und das Build-Tool herunter:

```
npm install
```

Sie können jetzt eine verpackte Version des SDK erstellen.

Die Erstellung von der Befehlszeile

Das Builder-Tool befindet sich hier: `dist-tools/browser-builder.js`. Führen Sie dieses Skript aus, indem Sie Folgendes eingeben:

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

Dieser Befehl erstellt die aws-sdk.js-Datei. Diese Datei ist unkomprimiert. Standardmäßig enthält dieses Paket nur den Standardsatz an Services.

Minimierte Build-Ausgabe

Um die Datenmenge im Netzwerk zu reduzieren, können JavaScript Dateien mithilfe eines Prozesses, der als Minimierung bezeichnet wird, komprimiert werden. Bei der Minimierung werden Kommentare, überflüssige Leerzeichen und andere Zeichen entfernt, die die Lesbarkeit unterstützen, jedoch keine Auswirkungen auf die Ausführung des Codes haben. Das Builder-Tool kann unkomprimierte oder minimiert Ausgaben generieren. Um Ihre Build-Ausgabe zu minimieren, legen Sie die Umgebungsvariable `MINIFY` fest:

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

Erstellen von bestimmten Services und API-Versionen

Sie können auswählen, welche Dienste im SDK erstellt werden sollen. Um die Services auszuwählen, geben Sie die Namen der Services durch Kommas getrennt als Parameter ein. Um beispielsweise nur Amazon S3 und Amazon EC2 zu erstellen, verwenden Sie den folgenden Befehl:

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

Indem Sie die Versionsbezeichnung nach dem Service-Namen hinzufügen, können Sie auch spezifische API-Versionen des Services erstellen. Um beispielsweise beide API-Versionen von Amazon DynamoDB zu erstellen, verwenden Sie den folgenden Befehl:

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

Service-IDs und API-Versionen sind in den dienstspezifischen Konfigurationsdateien unter https://github.com/aws/ /tree/master/apis verfügbar. aws-sdk-js

Erstellen aller Services

Sie können alle Services und API-Versionen erstellen, indem Sie den `all`-Parameter verwenden:

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

Erstellen bestimmter Services

Um die Auswahl der Services, die im Build eingeschlossen sein sollen, anzupassen, übergeben Sie die Umgebungsvariable `AWS_SERVICES` an den Browserify-Befehl, der die Liste der gewünschten Services enthält. Im folgenden Beispiel werden die Amazon EC2-, Amazon S3- und DynamoDB-Services erstellt.

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

Erstellen des SDK als Abhängigkeit mit Browserify

Node.js verfügt über einen modulbasierten Mechanismus, um Code und Funktionen von externen Entwicklern einzuschließen. Dieser modulare Ansatz wird bei der JavaScript Ausführung in Webbrowsern nicht nativ unterstützt. Jedoch steht Ihnen mithilfe des Browserify-Tools der Node.js-Modulansatz zur Verfügung und Sie können Module verwenden, die im Browser für Node.js geschrieben wurden. Browserify baut die Modulabhängigkeiten für ein Browserskript in einer einzigen, eigenständigen JavaScript Datei auf, die Sie im Browser verwenden können.

Sie können mithilfe von Browserify das SDK als Bibliotheksabhängigkeit für jedes Browser-Skript erstellen. Beispielsweise erfordert der folgende Node.js-Code das SDK:

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

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

Dieses Beispiel kann mithilfe von Browserify in eine Browser-kompatible Version kompiliert werden:

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

Die Anwendung, einschließlich der SDK-Abhängigkeiten, wird dann im Browser mithilfe von `browser-app.js` zur Verfügung gestellt.

Weitere Informationen über Browserify finden Sie auf der Browserify-Website.

Cross-Origin Resource Sharing (CORS)

Bei Cross-Origin Resource Sharing oder CORS handelt es sich um eine Sicherheitsfunktion der modernen Webbrowser. Es ermöglicht Webbrowsern zu verhandeln, welche Domänen Anforderungen von externen Websites oder Services senden können. CORS ist ein wichtiger Aspekt bei der Entwicklung von Browseranwendungen mit dem AWS SDK for JavaScript, da die meisten Anforderungen an Ressourcen an eine externe Domäne, wie z. B. der Endpunkt für einen Webservice, gesendet werden. Wenn Ihre JavaScript-Umgebung die CORS-Sicherheit erzwingt, müssen Sie CORS mit dem Service konfigurieren.

Ob die Freigabe von Ressourcen in einer ursprungsübergreifenden Anforderung zulässig ist, bestimmt CORS basierend auf:

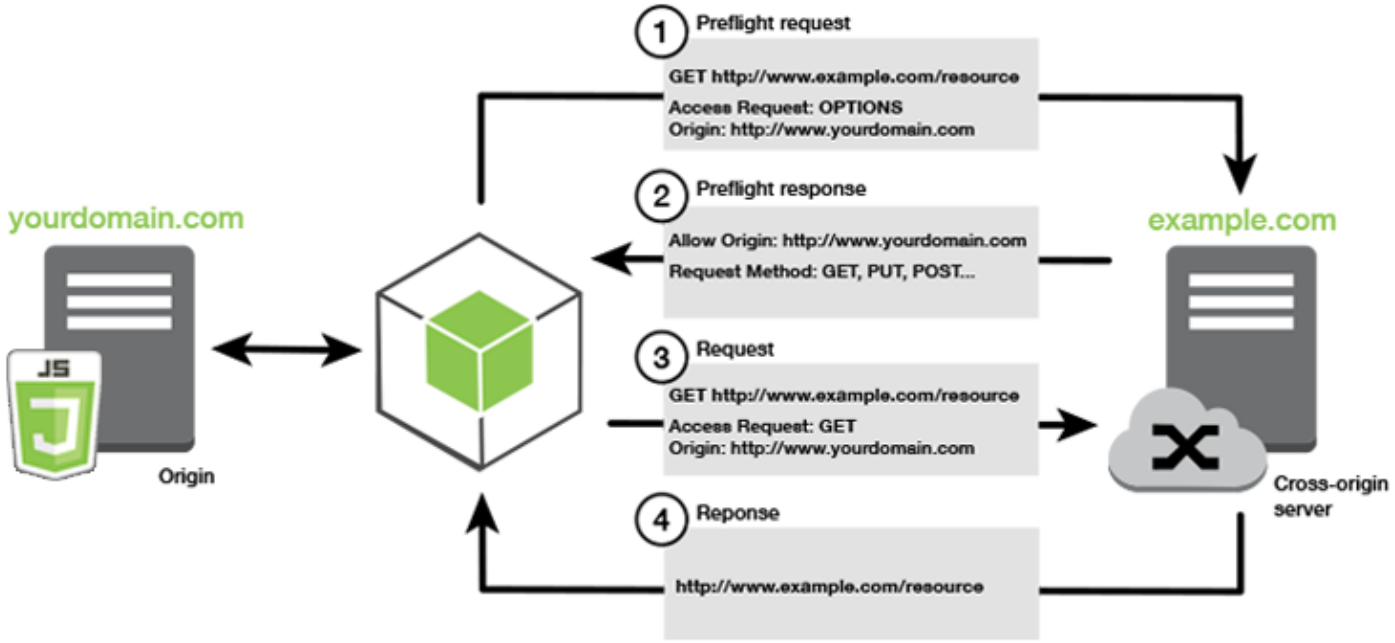
- Der spezifischen Domäne, die die Anforderung sendet
- Dem Typ der HTTP-Anforderung, die gesendet wird (GET, PUT, POST, DELETE usw.)

So funktioniert CORS

Im einfachsten Fall sendet Ihr Browser-Skript eine GET-Anforderung für eine Ressource von einem Server in einer anderen Domäne. Abhängig von der CORS-Konfiguration dieses Servers, wenn die Anforderung von einer Domäne stammt, die berechtigt ist, GET-Anforderungen abzusenden, reagiert der ursprungsübergreifende Server, indem er die angeforderte Ressource zurück gibt.

Wenn entweder die anfordernde Domäne oder der Typ der HTTP-Anforderung nicht autorisiert ist, wird die Anforderung abgelehnt. CORS jedoch ermöglicht ein Preflight der Anforderung, bevor sie tatsächlich abgesendet wird. In diesem Fall wird eine Preflight-Anforderung ausgeführt, in der

die `OPTIONS`-Zugriffsanforderungsoperation gesendet wird. Wenn die CORS-Konfiguration des ursprungsübergreifenden Servers Zugriff auf die anfordernde Domäne gewährt, sendet der Server eine Preflight-Antwort zurück, die alle HTTP-Anforderungstypen aufführt, die die anfordernde Domäne für die angeforderte Ressource vornehmen kann.



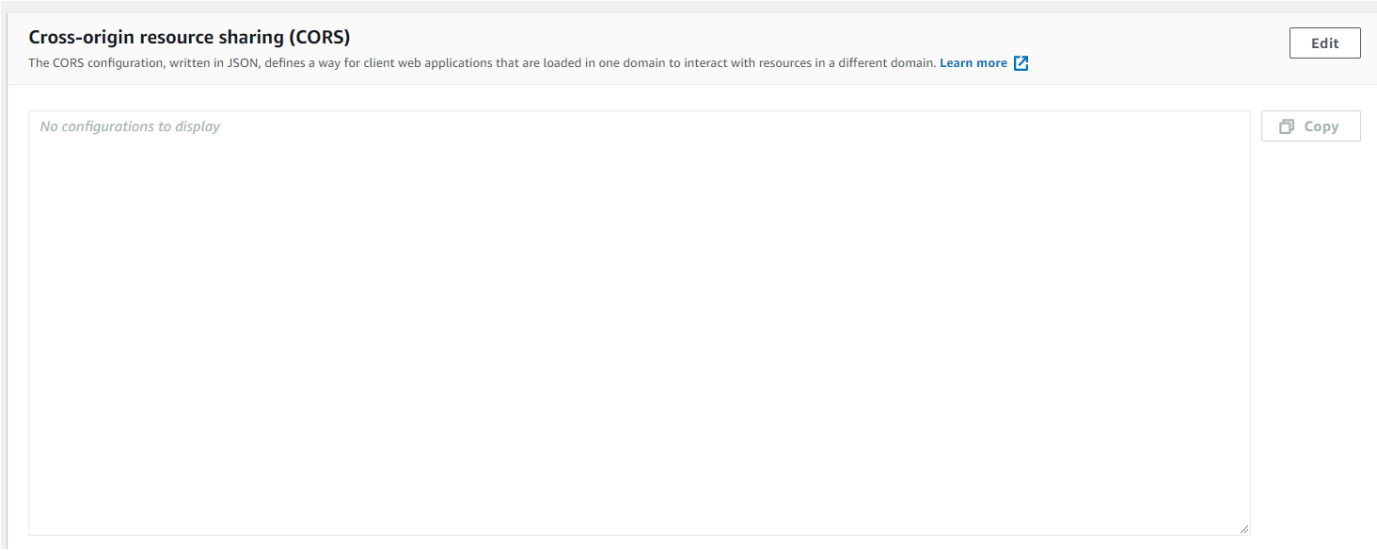
Ist die CORS-Konfiguration erforderlich

Amazon S3 S3-Buckets benötigen die CORS-Konfiguration, bevor Sie Operationen an ihnen ausführen können. In einigen JavaScript-Umgebungen wird CORS möglicherweise nicht erzwungen und daher ist das Konfigurieren von CORS nicht erforderlich. Beispiel: Wenn Sie Ihre Anwendung aus einem Amazon S3 S3-Bucket hosten und auf Ressourcen aus`*.s3.amazonaws.com`oder einem anderen spezifischen Endpunkt, Ihre Anforderungen greifen nicht auf eine externe Domäne zu. Aus diesem Grund benötigt diese Konfiguration kein CORS. In diesem Fall wird CORS immer noch für andere Services als Amazon S3 verwendet.

Konfigurieren von CORS für einen Amazon-S3-Bucket

Sie können einen Amazon-S3-Bucket zur Verwendung von CORS in der Amazon S3 S3-Konsole konfigurieren.

1. Wählen Sie in der Amazon-S3-Konsole den Bucket aus, den Sie bearbeiten möchten.
2. Wählen Sie dasBerechtigungenTab, und schimpfen Sie nach unten zumCross-Origin Resource Sharing (CORS)-Bedienfeld.



3. Klicken Sie aufBearbeiten, und geben Sie Ihre CORS-Konfiguration imCORS-Konfigurationseditor, dann wählenSaveaus.

Eine CORS-Konfiguration ist eine XML-Datei mit einer Reihe von Regeln innerhalb eines `<CORSRule>`. Eine Konfiguration kann bis zu 100 Regeln enthalten. Eine Regel wird durch eines der folgenden Tags definiert:

- `<AllowedOrigin>`, das Domänenursprünge angibt, denen Sie erlauben, domänenübergreifende Anforderungen zu senden.
- `<AllowedMethod>`, das einen Typ der Anforderung angibt, den Sie (GET, PUT, POST, DELETE, HEAD) in domänenübergreifenden Anforderungen erlauben.
- `<AllowedHeader>`, das die Header angibt, die in einer Preflight-Anforderung zulässig sind.

Beispielkonfigurationen finden Sie unterWie konfiguriere ich CORS für meinen Bucket?imAmazon Simple Storage Service — Benutzerhandbuchaus.

CORS-Konfigurationsbeispiel

Das folgende CORS-Konfigurationsbeispiel ermöglicht Benutzern das Anzeigen, Hinzufügen, Entfernen oder Aktualisieren von Objekten innerhalb eines Buckets aus der Domäne `example.org`, aber es wird empfohlen, dass Sie `<AllowedOrigin>` auf die Domäne Ihrer Website anwenden. Sie können `"*"` angeben, sodass jeder Ursprung zulässig ist.

> **Important**
>
> In der neuen S3-Konsole muss die CORS-Konfiguration JSON sein.

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"]
```

```
    }
]
```

Diese Konfiguration autorisiert den Benutzer nicht, Aktionen für den Bucket auszuführen. Sie aktiviert das Sicherheitsmodell des Browsers, um eine Anforderung an Amazon S3 zu gestatten. Berechtigungen müssen über Bucket-Berechtigungen oder IAM-Rollenberechtigungen konfiguriert werden.

Sie können`ExposeHeader`um das SDK Antwort-Header lesen zu lassen, die von Amazon S3 zurückgegeben werden. Wenn Sie beispielsweise den `ETag`-Header aus einem `PUT`- oder mehrteiligen Upload lesen möchten, müssen Sie das `ExposeHeader`-Tag in Ihrer Konfiguration einschließen, wie im vorherigen Beispiel gezeigt. Das SDK kann nur auf Header zugreifen, die über die CORS-Konfiguration bereitgestellt werden. Wenn Sie Metadaten für das Objekt festlegen, werden Werte als Header mit dem Präfix `x-amz-meta-` zurückgegeben, wie z. B. `x-amz-meta-my-custom-header`, und müssen auch auf die gleiche Weise bereitgestellt werden.

Bündeln von Anwendungen mit Webpack

Webanwendungen in Browser-Skripts oder die Verwendung von Codemodulen von Node.js erstellt Abhängigkeiten. Diese Codemodule können eigene Abhängigkeiten haben, was zu einer Sammlung von miteinander verbundenen Modulen führt, die Ihre Anwendung benötigt, um zu funktionieren. Zum Verwalten von Abhängigkeiten können Sie einen Modulpacker wie Webpack verwenden.

Der Webpack-Modulpacker analysiert den Code Ihrer Anwendung. Dabei sucht er nach `import`- oder `require`-Anweisungen, um Pakete zu erstellen, die alle erforderlichen Komponenten für Ihre Anwendung enthalten, damit sie problemlos über eine Webseite bereitgestellt werden können. Das SDK für JavaScript kann als eine der Abhängigkeiten in Webpack einbezogen werden, die im Ausgabepaket eingeschlossen werden sollen.

Weitere Informationen zu Webpack finden Sie auf der Seite zum Webpack-Modulpacker auf GitHub.

Installieren von Webpack

Damit Sie den Webpack-Modulpacker installieren können, müssen Sie zuerst npm, den Node.js-Paketmanager, installieren. Geben Sie den folgenden Befehl ein, um die Webpack-CLI und das JavaScript-Modul zu installieren.

```
npm install webpack
```

Möglicherweise müssen Sie auch ein Webpack-Plug-in installieren, um JSON-Dateien laden zu können. Geben Sie den folgenden Befehl ein, um das JSON-Loader-Plug-in zu installieren.

```
npm install json-loader
```

Konfigurieren von Webpack

Standardmäßig sucht Webpack nach einer JavaScript-Datei mit dem Namen `webpack.config.js` im Root-Verzeichnis Ihres Projekts. Diese Datei enthält Ihre Konfigurationsoptionen. Hier finden Sie ein Beispiel für eine `webpack.config.js`-Konfigurationsdatei.

```
// 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']
      }
    ]
  }
}
```

In diesem Beispiel ist `browser.js` als Eintrittspunkt angegeben. Der Eintrittspunkt ist die Datei, bei der Webpack die Suche nach importierten Modulen beginnt. Der Name für die Ausgabedatei ist als

`bundle.js` angegeben. Diese Ausgabedatei enthält den gesamten JavaScript-Code, der für die Ausführung der Anwendung erforderlich ist. Wenn der im Eintrittspunkt angegebene Code andere Module wie z. B. das SDK für JavaScript importiert oder benötigt, wird dieser Code gebündelt, ohne in der Konfiguration angegeben werden zu müssen.

Die Konfiguration im vorher installierten `json-loader`-Plug-in gibt Webpack an, wie JSON-Dateien importiert werden. Standardmäßig unterstützt Webpack nur JavaScript. Es verwendet jedoch Loader, um Unterstützung für den Import anderer Dateitypen hinzuzufügen. Da das SDK für JavaScript JSON-Dateien umfassend verwendet, gibt Webpack beim Generieren des Pakets einen Fehler aus, wenn`json-loader`ist nicht enthalten.

Ausführen von Webpack

Zum Erstellen einer Anwendung, die Webpack verwendet, fügen Sie Folgendes zum `scripts`-Objekt in Ihrer `package.json`-Datei hinzu.

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

Hier finden Sie eine beispielhafte `package.json`-Datei, die zeigt, wie Webpack hinzugefügt wird.

```
{
  "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"
  }
}
```

Geben Sie den folgenden Befehl ein, um die Anwendung zu erstellen.

```
npm run build
```

Der Webpack-Modulpacker generiert dann die JavaScript-Datei, die Sie im Root-Verzeichnis Ihres Projekts angegeben haben.

Verwenden des Webpack-Pakets

Um das Paket in einem Browser-Skript zu verwenden, können Sie es mithilfe eines `<script>`-Tags integrieren, wie im folgenden Beispiel gezeigt.

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

Importieren von einzelnen Services

Einer der Vorteile von Webpack besteht darin, dass es die Abhängigkeiten in Ihrem Code analysiert und nur den für Ihre Anwendung erforderlichen Code bündelt. Wenn Sie das SDK für JavaScript verwenden, kann das Bündeln nur der von Ihrer Anwendung tatsächlich genutzten Teile des SDKs die Größe der Webpack-Ausgabe deutlich reduzieren.

Betrachten Sie das folgende Codebeispiel zum Erstellen eines Amazon S3-Service-Objekts.

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

Mit der `require()`-Funktion wird das gesamte SDK angegeben. Ein mit diesem Code generiertes Webpack-Paket umfasst das vollständige SDK. Dieses ist jedoch nicht erforderlich, wenn nur die

Amazon S3 S3--Client-Klasse verwendet wird. Die Größe des Pakets ist dann wesentlich kleiner, wenn nur der -Anteil des SDKs aufgenommen wird, der für den Amazon S3-Service erforderlich ist. Auch die Konfiguration erfordert nicht das vollständige SDK, da Sie die Konfigurationsdaten im Amazon S3-Service-Objekt festlegen können.

Nachfolgend sehen Sie, wie derselbe Code aussieht, wenn er nur den Amazon S3-Teil des SDKs enthält.

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

Bündeln für Node.js

Sie können mit Webpack Pakete generieren, die in Node.js ausgeführt werden, indem Sie es als Ziel in der Konfiguration angeben.

```
target: "node"
```

Dies ist nützlich, wenn eine Node.js-Anwendung in einer Umgebung ausgeführt wird, in der Speicherplatz begrenzt ist. Hier sehen Sie ein Beispiel für eine `webpack.config.js`-Konfiguration mit der Angabe von Node.js als Ausgabeziel.

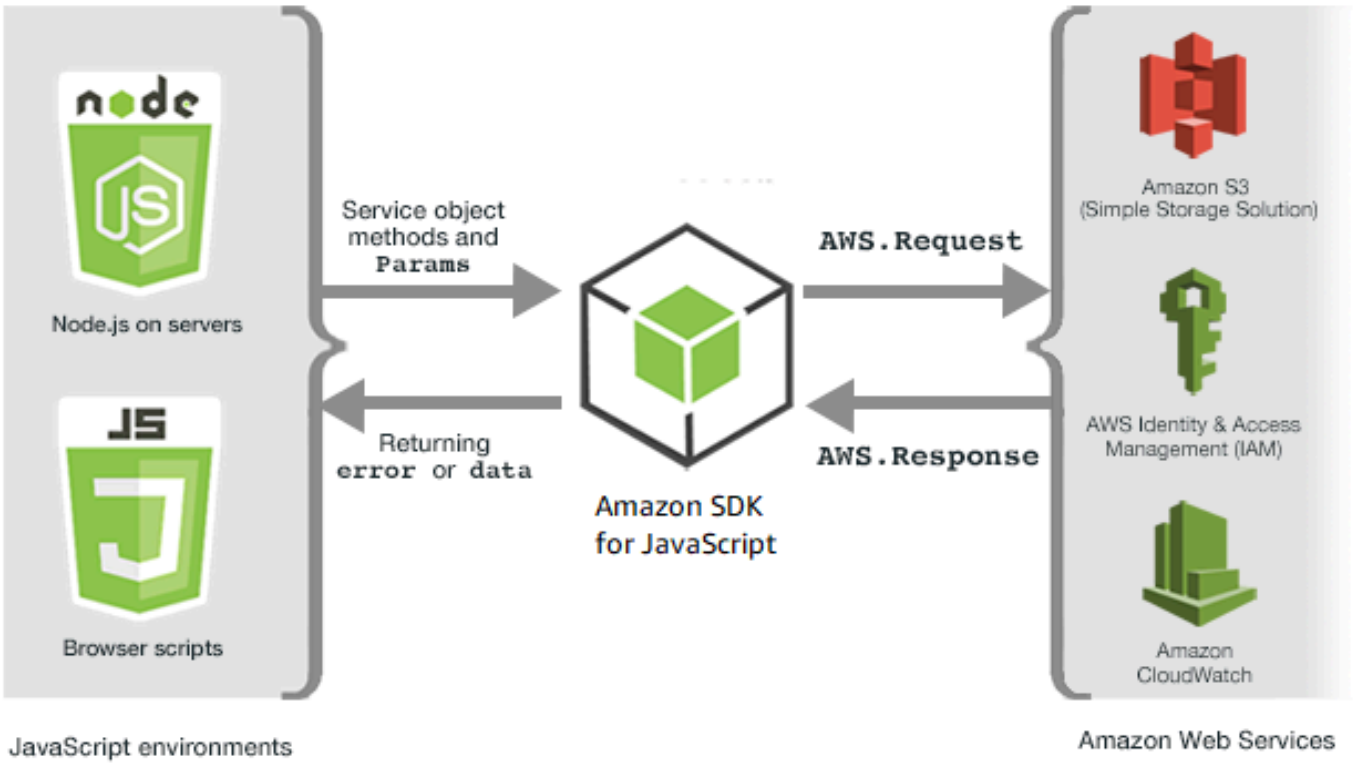
```
// Import path for resolving file paths
var path = require('path');
module.exports = {
  // Specify the entry point for our app
  entry: [
    path.join(__dirname, 'node.js')
  ],
  // Specify the output file containing our bundled code
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
```

```
  // 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']
      }
    ]
  }
}
```

Arbeiten mit Diensten im SDK für JavaScript

Das AWS SDK for JavaScript bietet Zugriff auf Services, die es durch eine Sammlung von Client-Klassen unterstützt. Von diesen Client-Klassen erstellen Sie Service-Schnittstellenobjekte, die allgemein als Service-Objekte bezeichnet werden. Jeder unterstützte AWS-Service verfügt über mindestens eine Client-Klasse, die Low-Level-APIs für die Verwendung von Servicefunktionen und -ressourcen bietet. Beispielsweise sind Amazon DynamoDB DynamoDB-APIs über die `AWS.DynamoDB` Klasse verfügbar.

Die über das SDK bereitgestellten Dienste JavaScript folgen dem Anfrage-Antwort-Muster, um Nachrichten mit aufrufenden Anwendungen auszutauschen. In diesem Muster sendet der Code, der einen Service aufruft, eine HTTP-/HTTPS-Anforderung an einen Endpunkt für den Service. Die Anforderung enthält die erforderlichen Parameter für einen erfolgreichen Aufruf der spezifischen Funktion. Der Service, der aufgerufen wird, generiert eine Antwort, die an den Anforderer zurückgesendet wird. Die Antwort enthält Daten, wenn die Operation erfolgreich war, oder Fehlerinformationen, wenn sie nicht erfolgreich war.



Das Aufrufen eines AWS Dienstes umfasst den gesamten Anforderungs- und Antwortzyklus eines Vorgangs an einem Dienstobjekt, einschließlich aller Wiederholungsversuche. Eine Anforderung wird durch das `AWS.Request`-Objekt im SDK gekapselt. Die Antwort wird im SDK durch das

`AWS.Response` Objekt gekapselt, das dem Anforderer über eine von mehreren Techniken zur Verfügung gestellt wird, z. B. durch eine Callback-Funktion oder eine Zusage. JavaScript

Themen

Erstellen und Aufrufen von Service-Objekten

Die JavaScript API unterstützt die meisten verfügbaren AWS Dienste. Jede Serviceklasse in der JavaScript API bietet Zugriff auf jeden API-Aufruf in ihrem Dienst. Weitere Informationen zu Serviceklassen, Vorgängen und Parametern in der JavaScript API finden Sie in der API-Referenz.

Wenn Sie das SDK in Node.js verwenden, fügen Sie Ihrer Anwendung das SDK-Paket mit `require` hinzu, das Unterstützung für alle aktuellen Services bietet.

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

Wenn Sie das SDK mit einem Browser verwenden JavaScript, laden Sie das SDK-Paket mithilfe des von AWS gehosteten SDK-Pakets in Ihre Browserskripte. Fügen Sie das folgende `<script>` Element hinzu, um das SDK-Pakezt zu laden:

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

Die aktuelle SDK_VERSION_NUMBER finden Sie in der API-Referenz für das SDK im API-Referenzhandbuch. JavaScript AWS SDK for JavaScript

Das standardmäßige gehostete SDK-Paket bietet Unterstützung für eine Teilmenge der verfügbaren Dienste. AWS Eine Liste der Standard-Services im gehosteten SDK-Paket für den Browser finden Sie im Abschnitt über Unterstützte Services in der API-Referenz. Sie können das SDK mit anderen Services verwenden, wenn die CORS-Sicherheitsüberprüfung deaktiviert ist. In diesem Fall können Sie eine benutzerdefinierte Version des SDKs erstellen, um die von Ihnen benötigten zusätzlichen

Services einzuschließen. Weitere Informationen zum Erstellen einer benutzerdefinierten Version des SDKs finden Sie unter Erstellen des SDK für Browser.

Laden einzelner Services mit der require-Funktion

Wenn Sie das SDK für JavaScript wie zuvor beschrieben benötigen, wird das gesamte SDK in Ihren Code aufgenommen. Sie können aber auch nur die einzelnen Services mit der require-Funktion laden, die von Ihrem Code verwendet werden. Betrachten Sie den folgenden Code, der verwendet wurde, um ein Amazon S3 S3-Serviceobjekt zu erstellen.

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

Im vorherigen Beispiel gibt die `require`-Funktion das gesamte SDK an. Die Menge an Code, die über das Netzwerk übertragen werden muss, sowie der Speicheraufwand Ihres Codes wären erheblich geringer, wenn nur der Teil des SDK enthalten wäre, den Sie für den Amazon S3 S3-Service benötigen. Um einen einzelnen Service zu laden, rufen Sie die `require`-Funktion wie beschrieben auf, einschließlich des Service-Konstruktors in Kleinbuchstaben.

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

So sieht der Code zum Erstellen des vorherigen Amazon S3 S3-Serviceobjekts aus, wenn er nur den Amazon S3 S3-Teil des SDK enthält.

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

Sie können immer noch auf den globalen AWS Namespace zugreifen, ohne dass jeder Service damit verbunden ist.

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

Dies ist eine nützliche Technik, wenn dieselbe Konfiguration auf mehrere individuelle Services angewendet werden soll, um z. B. dieselben Anmeldeinformationen allen Services bereitzustellen. Das Laden einzelner Services mit der require-Funktion sollte die Ladezeit und die Speichernutzung in Node.js verringern. Wenn Sie diesen Vorgang mit einem Bündelungs-Tool wie z. B. Browserify oder Webpack abgeschlossen haben, weist das SDK nur einen Bruchteil der vollständigen Größe auf, weil nur einzelne Service-Ergebnisse mit der require-Funktion geladen wurden. Dies hilft in Umgebungen mit begrenztem Speicher oder Festplattenspeicher, wie z. B. bei einem IoT-Gerät oder bei einer Lambda-Funktion.

Erstellen von Service-Objekten

Um über die JavaScript API auf Servicefunktionen zuzugreifen, erstellen Sie zunächst ein Serviceobjekt, über das Sie auf eine Reihe von Funktionen zugreifen, die von der zugrunde liegenden Clientklasse bereitgestellt werden. In der Regel wird für jeden Service eine Client-Klasse bereitgestellt. Manche Services verteilen den Zugriff auf ihre Funktionen jedoch auf mehrere Client-Klassen.

Zum Verwenden einer Funktion müssen Sie eine Instance der Klasse erstellen, die Zugriff auf diese Funktion bietet. Das folgende Beispiel zeigt die Erstellung eines Serviceobjekts für DynamoDB aus der `AWS.DynamoDB` Client-Klasse.

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

Standardmäßig wird ein Service-Objekt mit den globalen Einstellungen konfiguriert, mit denen auch das SDK konfiguriert wird. Sie können jedoch ein Service-Objekt mit Laufzeit-Konfigurationsdaten konfigurieren, die für das Service-Objekt spezifisch sind. Service-spezifische Konfigurationsdaten werden nach den globalen Konfigurationseinstellungen angewendet.

Im folgenden Beispiel wird ein Amazon EC2-Serviceobjekt mit einer Konfiguration für eine bestimmte Region erstellt, verwendet aber ansonsten die globale Konfiguration.

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

Es werden nicht nur servicespezifische Konfigurationen unterstützt, die auf einzelne Service-Objekte angewendet werden. Sie können auch servicespezifische Konfigurationen auf alle neu erstellten Service-Objekte einer bestimmten Klasse anwenden. Um beispielsweise alle Serviceobjekte, die mit der Amazon EC2 EC2-Klasse erstellt wurden, für die Verwendung der Region USA West (Oregon) (`us-west-2`) zu konfigurieren, fügen Sie dem `AWS.config` globalen Konfigurationsobjekt Folgendes hinzu.

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

Sperren der API-Version eines Service-Objekts

Sie können ein Service-Objekt fest an eine bestimmte API-Version eines Service binden, indem Sie die `apiVersion`-Option beim Erstellen des Objekts angeben. Im folgenden Beispiel wird ein DynamoDB-Dienstobjekt erstellt, das an eine bestimmte API-Version gebunden ist.

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

Weitere Informationen zum Sperren der API-Version eines Service-Objekts finden Sie unter Schützen der API-Versionen.

Angeben von Service-Objektparametern

Wenn Sie eine Methode eines Service-Objekts aufrufen, übergeben Sie die JSON-Parameter wie für die API erforderlich. Um beispielsweise in Amazon S3 ein Objekt für einen bestimmten Bucket und Schlüssel abzurufen, übergeben Sie die folgenden Parameter an die `getObject` Methode. Weitere Informationen zum Übergeben von JSON-Parametern finden Sie unter Arbeiten mit JSON.

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

Weitere Informationen zu Amazon S3 S3-Parametern finden Sie `Class: AWS.S3`in der API-Referenz.

Darüber hinaus können Sie Werte an einzelne Parameter binden, wenn ein Service-Objekt mit dem `params`-Parameter erstellt wird. Der Wert des `params`-Parameters von Service-Objekten ist eine Zuordnung, mit der mindestens ein durch das Service-Objekt definierter Parameterwert angegeben wird. Das folgende Beispiel zeigt den `Bucket` Parameter eines Amazon S3 S3-Serviceobjekts, das an einen Bucket mit dem Namen gebunden ist`myBucket`.

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

Indem das Service-Objekt an einen Bucket gebunden wird, behandelt das `s3bucket`-Service-Objekt den `myBucket`-Parameterwert als Standardwert, der nicht länger für nachfolgende Operationen angegeben werden muss. Alle gebundenen Parameterwerte werden ignoriert, wenn das Objekt für Operationen verwendet wird, bei denen der Parameterwert nicht anwendbar ist. Sie können diesen gebundenen Parameter beim Aufrufen des Service-Objekts überschreiben, indem Sie einen neuen Wert angeben.

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

Detaillierte Informationen zu den verfügbaren Parametern für die einzelnen Methoden finden Sie in der API-Referenz.

Protokollieren von AWS SDK for JavaScript-Aufrufen

Der AWS SDK for JavaScript ist mit einem integrierten Logger ausgestattet, sodass Sie API-Aufrufe protokollieren können, die Sie mit dem SDK für JavaScript tätigen.

Fügen Sie Ihrem Code die folgende Anweisung hinzu, um den Logger zu aktivieren und die Protokolleinträge in der Konsole zu drucken.

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

Hier finden Sie ein Beispiel für die Protokollausgabe.

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

Verwenden eines Drittanbieter-Loggers

Sie können auch einen Drittanbieter-Logger verwenden, sofern dieser über `log()`- oder `write()`-Operationen verfügt, um Daten in eine Protokolldatei oder auf einen Server zu schreiben. Sie müssen

Ihren benutzerdefinierten Logger gemäß den Anweisungen installieren und einrichten, bevor Sie ihn mit dem SDK für verwenden können. JavaScript

Ein Logger, den Sie entweder in Browser-Skripts oder in Node.js nutzen können, ist logplease. In Node.js können Sie logplease so konfigurieren, dass Protokolleinträge in einer Protokolldatei geschrieben werden. Sie können ihn auch mit webpack verwenden.

Legen Sie bei Verwendung eines Drittanbieter-Loggers alle Optionen fest, bevor Sie ihn `AWS.Config.logger` zuweisen. Der folgende Beispiel-Code gibt eine externe Protokolldatei an und legt die Protokollebene für logplease fest.

```
// 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;
```

Weitere Informationen zu Logplease finden Sie im Logplease Simple JavaScript Logger on. GitHub

Asynchrones Aufrufen von Services

Alle über das SDK ausgeführten Anforderungen sind asynchron. Dies ist wichtig, wenn Sie Browserskripte schreiben. JavaScript Die Ausführung in einem Webbrowser hat normalerweise nur einen einzigen Ausführungsthread. Nachdem Sie einen asynchronen Aufruf an einen AWS-Service durchgeführt haben, wird das Browser-Skript weiter ausgeführt, das dabei möglicherweise versucht, Code auszuführen, der von diesem asynchronen Ergebnis abhängig ist, bevor es zurückgegeben wird.

Zur Durchführung asynchroner Aufrufe an einen AWS-Service gehört die Verwaltung solcher Aufrufe, sodass Ihr Code nicht versucht, Daten zu verwenden, bevor diese verfügbar sind. Die Themen in diesem Abschnitt erklären, wie wichtig es ist, asynchrone Aufrufe zu verwalten, und erläutern die verschiedenen Verwaltungstechniken im Detail.

Themen

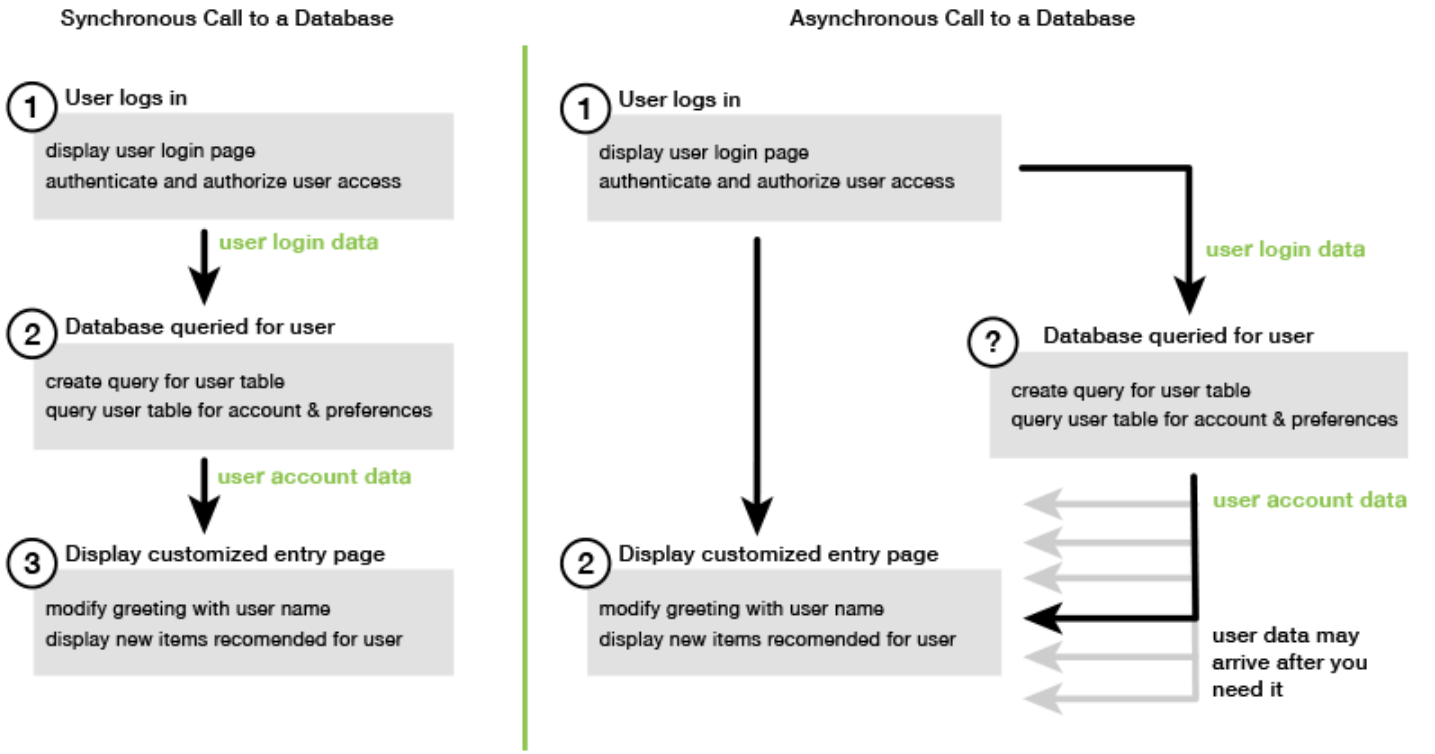
Verwalten von asynchronen Aufrufen

Beispiel: Die Startseite einer E-Commerce-Website bietet wiederkehrenden Kunden die Möglichkeit, sich anzumelden. Die Kunden, die sich anmelden, profitieren u. a. davon, dass sich die Website nach der Anmeldung an ihre besonderen Präferenzen anpasst. Dafür ist Folgendes erforderlich:

1. Der Kunde muss sich anmelden und mit seinen Anmeldedaten validiert werden.
2. Die Präferenzen des Kunden werden von einer Kundendatenbank angefordert.
3. Die Datenbank stellt die Präferenzen des Kunden bereit, mit denen die Website angepasst wird, bevor die Seite geladen wird.

Wenn diese Aufgaben synchron ausgeführt werden, muss jede abgeschlossen sein, bevor die nächste starten kann. Die Webseite kann nicht vollständig geladen werden, bis die Präferenzen des Kunden von der Datenbank zurückgegeben wurden. Nachdem die Datenbankabfrage an den Server gesendet wurde, kann der Empfang der Kundendaten jedoch aufgrund von Netzwerkengpässen, besonders hohem Datenverkehr oder einer schlechten Verbindung eines Mobilgeräts sich verzögern oder sogar fehlschlagen.

Rufen Sie die Datenbank asynchron auf, um zu verhindern, dass die Website unter solchen Bedingungen einfriert. Nachdem der Datenbankaufruf ausgeführt und Ihre asynchrone Anforderung gesendet wurde, wird Ihr Code wie erwartet weiter ausgeführt. Wenn Sie die Antwort eines asynchronen Aufrufs nicht ordnungsgemäß verwalten, versucht Ihr Code möglicherweise im Fall, dass diese Daten noch nicht verfügbar sind, Informationen zu verwenden, die er von der Datenbank zurückerwartet.



Verwenden einer anonymen Callback-Funktion

Jede Service-Objektmethode, die ein `AWS.Request`-Objekt erstellt, kann eine anonyme Callback-Funktion als letzten Parameter annehmen. Die Signatur dieser Callback-Funktion lautet:

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

Diese Callback-Funktion wird ausgeführt, wenn entweder eine erfolgreiche Antwort oder Fehlerdaten zurückgegeben werden. Wenn der Methodenaufruf erfolgreich war, wird der Callback-Funktion der Inhalt der Antwort im `data`-Parameter bereitgestellt. Wenn der Aufruf nicht erfolgreich war, werden die Details über den Fehler im `error`-Parameter angegeben.

In der Regel prüft der Code innerhalb der Callback-Funktion, ob ein Fehler vorliegt, den er verarbeitet, wenn ein Fehler zurückgegeben wird. Wenn kein Fehler zurückgegeben wird, ruft der Code die Antwortdaten vom `data`-Parameter ab. Die grundlegende Form der Callback-Funktion sieht wie folgt aus.

```
function(error, data) {
    if (error) {
```

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

Im vorherigen Beispiel werden die Details zum Fehler oder die zurückgegebenen Daten in der Konsole protokolliert. Hier finden Sie ein Beispiel für eine Callback-Funktion, die als Teil des Aufrufs einer Methode in einem Service-Objekt zurückgegeben wird.

```
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
  }
});
```

Zugreifen auf die Anforderungs- und Antwortobjekte

Innerhalb der Callback-Funktion `this` bezieht sich das JavaScript Schlüsselwort auf das zugrunde liegende `AWS.Response` Objekt für die meisten Dienste. Im folgenden Beispiel wird die `httpResponse`-Eigenschaft eines `AWS.Response`-Objekts innerhalb einer Callback-Funktion für die Protokollierung der unformatierten Antwortdaten und -Header verwendet, um das Debugging zu erleichtern.

```
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
  }
});
```

Da das `AWS.Response`-Objekt über eine `Request`-Eigenschaft verfügt, die das `AWS.Request`-Objekt enthält, das vom ursprünglichen Methodenaufruf gesendet wurde, können Sie außerdem auf die Details der Anforderung zugreifen.

Verwenden eines Anforderungsobjektereignis-Listeners

Wenn Sie keine anonyme Callback-Funktion erstellen und als Parameter übergeben, wenn Sie eine Service-Objektmethode aufrufen, generiert der Methodenaufruf ein `AWS.Request`-Objekt, das mit dessen `send`-Methode manuell gesendet werden muss.

Zum Verarbeiten der Antwort müssen Sie einen Ereignis-Listener für das `AWS.Request`-Objekt erstellen, um eine Callback-Funktion für den Methodenaufruf zu registrieren. Das folgende Beispiel zeigt, wie das `AWS.Request`-Objekt zum Aufrufen einer Service-Objektmethode und der Ereignis-Listener für die erfolgreiche Rückgabe erstellt werden.

```
// 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();
```

Nachdem die `send`-Methode im `AWS.Request`-Objekt aufgerufen wurde, wird der Ereignis-Handler ausgeführt, wenn das Service-Objekt ein `AWS.Response`-Objekt erhält.

Weitere Informationen zu dem `AWS.Request` Objekt finden Sie `Class: AWS.Request`in der API-Referenz. Weitere Informationen über das `AWS.Response` Objekt finden Sie unter Verwenden des Response-Objekts oder `Class: AWS.Response`in der API-Referenz.

Verketten von mehreren Callbacks

Sie können mehrere Callbacks für jedes Anforderungsobjekt registrieren. Mehrere Callbacks können für verschiedene Ereignisse oder dasselbe Ereignis registriert werden. Sie können Callbacks auch verketten, wie im folgenden Beispiel gezeigt wird.

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

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

Anforderungsobjekt-Fertigstellungsereignis

Das `AWS.Request`-Objekt löst diese Fertigstellungsereignisse basierend auf der Antwort der einzelnen Service-Operationsmethoden aus:

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

Sie können eine Callback-Funktion als Reaktion auf jedes dieser Ereignisse registrieren. Eine vollständige Liste aller Ereignisse mit Anforderungsobjekten finden Sie `Class: AWS.Request` in der API-Referenz.

Das Erfolgsereignis

Das `success`-Ereignis wird ausgelöst, wenn eine erfolgreiche Antwort vom Service-Objekt erhalten wird. Hier sehen Sie, wie Sie eine Callback-Funktion für dieses Ereignis registrieren.

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

Die Antwort enthält eine `data`-Eigenschaft mit den serialisierten Antwortdaten vom Service. Zum Beispiel der folgende Aufruf der `listBuckets` Methode des Amazon S3-Serviceobjekts

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

gibt die Antwort zurück. Anschließend wird der folgende Inhalt der `data`-Eigenschaft an die Konsole ausgegeben.

```
{ Owner: { ID: '...', DisplayName: '...' },
```

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

Das error-Ereignis

Das `error`-Ereignis wird ausgelöst, wenn eine Fehlerantwort vom Service-Objekt erhalten wird. Hier sehen Sie, wie Sie eine Callback-Funktion für dieses Ereignis registrieren.

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

Wenn das `error`-Ereignis ausgelöst wird, ist der Wert der `data`-Eigenschaft der Antwort `null` und die `error`-Eigenschaft enthält die Fehlerdaten. Das zugehörige `error`-Objekt wird als erster Parameter an die registrierte Callback-Funktion übergeben. Der folgende Code beispielsweise:

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

gibt den Fehler zurück. Anschließend werden die folgenden Fehlerdaten an die Konsole ausgegeben.

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

Das complete-Ereignis

Das `complete`-Ereignis wird ausgelöst, wenn ein Service-Objektaufruf unabhängig davon, ob der Aufruf erfolgreich war oder nicht, abgeschlossen wurde. Hier sehen Sie, wie Sie eine Callback-Funktion für dieses Ereignis registrieren.

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

Verwenden Sie den `complete`-Ereignis-Callback, um die einzelnen Anforderungsbereinigungen zu verarbeiten, die unabhängig von Erfolg oder Fehlern ausgeführt werden müssen. Wenn Sie Antwortdaten in einer Callback-Funktion für das `complete`-Ereignis verwenden, überprüfen Sie

zuerst die Eigenschaften `response.data` oder `response.error`, bevor Sie versuchen, auf eine der beiden zuzugreifen. Dies wird in folgendem Beispiel gezeigt.

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

HTTP-Ereignisse für Anforderungsobjekte

Das `AWS.Request`-Objekt löst die folgenden HTTP-Ereignisse basierend auf der Antwort der einzelnen Service-Operationsmethoden aus:

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

Sie können eine Callback-Funktion als Reaktion auf jedes dieser Ereignisse registrieren. Eine vollständige Liste aller Ereignisse mit Anforderungsobjekten finden Sie `Class: AWS.Request` in der API-Referenz.

Das httpHeaders-Ereignis

Das `httpHeaders`-Ereignis wird ausgelöst, wenn Header vom Remote-Server gesendet werden. Hier sehen Sie, wie Sie eine Callback-Funktion für dieses Ereignis registrieren.

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

Der `statusCode`-Parameter der Callback-Funktion ist der HTTP-Statuscode. Der `headers`-Parameter enthält die Antwort-Header.

Das httpData-Ereignis

Das `httpData`-Ereignis wird ausgelöst, um Antwortdatenpakete vom Service zu streamen. Hier sehen Sie, wie Sie eine Callback-Funktion für dieses Ereignis registrieren.

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

Dieses Ereignis wird in der Regel für den Empfang großer Antwortblöcke verwendet, wenn das Laden der gesamten Antwort in den Arbeitsspeicher nicht praktikabel ist. Dieses Ereignis verfügt über einen zusätzlichen `chunk`-Parameter, der einen Teil der eigentlichen Daten vom Server enthält.

Wenn Sie einen Callback für das `httpData`-Ereignis registrieren, enthält die `data`-Eigenschaft der Antwort die gesamte serialisierte Ausgabe für die Anforderung. Sie müssen den standardmäßigen `httpData`-Listener entfernen, wenn Sie nicht über den zusätzlichen Analyse- und Speichermehraufwand für die integrierten Handler verfügen.

Die httpUploadProgress und httpDownloadProgress Ereignisse

Das `httpUploadProgress`-Ereignis wird ausgelöst, wenn die HTTP-Anforderung mehr Daten hochgeladen hat. Entsprechend wird das `httpDownloadProgress`-Ereignis ausgelöst, wenn die HTTP-Anforderung mehr Daten heruntergeladen hat. Hier sehen Sie, wie Sie eine Callback-Funktion für diese Ereignisse registrieren.

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

Der `progress`-Parameter der Callback-Funktion enthält ein Objekt mit den geladenen und gesamten Bytes der Anforderung.

Das httpError-Ereignis

Das `httpError`-Ereignis wird ausgelöst, wenn die HTTP-Anforderung fehlschlägt. Hier sehen Sie, wie Sie eine Callback-Funktion für dieses Ereignis registrieren.

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

Der `error`-Parameter der Callback-Funktion enthält die Fehler, die ausgelöst wurden.

Das httpDone-Ereignis

Das `httpDone`-Ereignis wird ausgelöst, wenn der Server das Senden von Daten abgeschlossen hat. Hier sehen Sie, wie Sie eine Callback-Funktion für dieses Ereignis registrieren.

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

Verwenden von async/await

Sie können das `async/await` Muster in Ihren Aufrufen an die verwendenAWS SDK for JavaScript. Die meisten Funktionen, die einen Rückruf annehmen, geben kein Versprechen zurück. Da Sie nur `await` Funktionen verwenden, die ein Versprechen zurückgeben, müssen Sie, um das `async/await` Muster zu verwenden, die `.promise()` Methode bis zum Ende Ihres Aufrufs verketten und den Callback entfernen.

Das folgende Beispiel verwendet async/await, um all Ihre Amazon DynamoDB-Tabellen in aufzulisten. `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**
>
> Nicht alle Browser unterstützen Async/Await. Eine Liste von Browsern mit Async/Await-Unterstützung finden Sie unter Async-Funktionen.

JavaScript Promises verwenden

Mit der `AWS.Request.promise`-Methode haben Sie die Möglichkeit, eine Service-Operation aufzurufen und asynchrone Flows zu verwalten, anstatt Callbacks zu verwenden. In Node.js und Browser-Skripts wird ein `AWS.Request`-Objekt zurückgegeben, wenn eine Service-Operation ohne eine Callback-Funktion aufgerufen wird. Sie können die `send`-Methode der Anforderung aufrufen, um den Service aufzurufen.

`AWS.Request.promise` startet jedoch sofort den Service-Aufruf und gibt ein Promis zurück. Dieses wird entweder mit der `data`-Eigenschaft der Antwort erfüllt oder mit der `error`-Eigenschaft der Antwort abgelehnt.

```
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 */
  }
);
```

Das nächste Beispiel gibt ein Promis zurück, das mit einem `data`-Objekt erfüllt oder mit einem `error`-Objekt abgelehnt wird. Beim Verwenden von Promises ist ein einzelner Callback nicht dafür zuständig, Fehler zu erkennen. Stattdessen wird der richtige Callback basierend auf dem Erfolg oder Misserfolg einer Anforderung aufgerufen.

```
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);
});
```

Koordinieren mehrerer Promises

In einigen Fällen muss Ihr Code mehrere asynchrone Aufrufe ausführen, die nur dann eine Aktion erfordern, wenn alle erfolgreich zurückgegeben wurden. Wenn Sie diese einzelnen asynchronen Methodenaufrufe mit Promises verwalten, können Sie ein zusätzliches Promise erstellen, das die `all`-Methode verwendet. Diese Methode erfüllt dieses übergeordnete Promise-Objekt, falls und wenn das an die Methode übergebene Array der Promises erfüllt ist. Die Callback-Funktion wird als Array von Werten der Promises übergeben, die an die `all`-Methode übergeben werden.

Im folgenden Beispiel muss eine AWS Lambda Funktion drei asynchrone Aufrufe an Amazon DynamoDB tätigen, kann aber erst abgeschlossen werden, nachdem die Zusagen für jeden Aufruf erfüllt wurden.

```
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);
});
```

Browser- und Node.js-Unterstützung für Promises

Die Support für native JavaScript Promises (ECMAScript 2015) hängt von der JavaScript Engine und Version ab, in der Ihr Code ausgeführt wird. Informationen zur Unterstützung von Promises in jeder Umgebung JavaScript , in der Ihr Code ausgeführt werden muss, finden Sie in der ECMAScript-Kompatibilitätstabelle unter. GitHub

Verwenden anderer Promise-Implementierungen

Zusätzlich zur nativen Promise-Implementierung in ECMAScript 2015 können Sie auch Promise-Bibliotheken von Drittanbietern verwenden. Dazu gehören:

- Bluebird
- RSVP
- Q

Diese optionalen Promise-Bibliotheken können nützlich sein, wenn Sie Ihren Code in Umgebungen ausführen müssen, die native Promise-Implementierungen in ECMAScript 5 und ECMAScript 2015 nicht unterstützen.

Wenn Sie die Promise-Bibliothek eines Drittanbieters verwenden möchten, legen Sie eine Promise-Abhängigkeit im SDK fest, indem Sie die `setPromisesDependency`-Methode des globalen Konfigurationsobjekts aufrufen. Stellen Sie in Browser-Skripts sicher, dass die Promise-Drittanbieterbibliothek vor dem SDK geladen wird. Im folgenden Beispiel wird das SDK so konfiguriert, dass es die Implementierung in der Bluebird-Promise-Bibliothek verwendet.

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

Um wieder die native Promise-Implementierung der JavaScript Engine zu verwenden, rufen Sie `setPromisesDependency` erneut auf und übergeben Sie `null` statt eines Bibliotheksnamens einen.

Verwenden des Response-Objekts

Nachdem eine Service-Objektmethode aufgerufen wurde, gibt diese ein `AWS.Response`-Objekt zurück, indem es an Ihre Callback-Funktion übergeben wird. Sie greifen auf den Inhalt der Antwort über die Eigenschaften des `AWS.Response`-Objekts zu. Für den Zugriff auf den Inhalt der Antwort gibt es zwei Eigenschaften des `AWS.Response`-Objekts:

- `data` Eigenschaft
- `error` Eigenschaft

Wenn Sie den standardmäßigen Callback-Mechanismus verwenden, werden diese beiden Eigenschaften als Parameter in der anonymen Callback-Funktion bereitgestellt, wie das folgende Beispiel zeigt.

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

Zugreifen auf im Response-Objekt zurückgegebene Daten

Die `data`-Eigenschaft des `AWS.Response`-Objekts enthält die serialisierten Daten, die von der Service-Anforderung zurückgegeben werden. Wenn die Anforderung erfolgreich ist, enthält die `data`-Eigenschaft ein Objekt mit einer Zuordnung zu den zurückgegebenen Daten. Die `data`-Eigenschaft kann Null sein, wenn ein Fehler auftritt.

Hier ist ein Beispiel für den Aufruf der `getItem` Methode einer DynamoDB-Tabelle, um den Dateinamen einer Bilddatei abzurufen, die als Teil eines Spiels verwendet werden soll.

```
// 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();
```

In diesem Beispiel ist die DynamoDB-Tabelle eine Suche nach Bildern, die die Ergebnisse eines Spielautomaten-Pulls zeigen, wie durch die Parameter in angegeben. `slotParams`

Bei einem erfolgreichen Aufruf der `getItem` Methode enthält die `data` Eigenschaft des `AWS.Response` Objekts ein von DynamoDB zurückgegebenes `Item` Objekt. Auf die zurückgegebenen Daten wird gemäß dem `ProjectionExpression`-Parameter der Anforderung zugegriffen, in diesem Fall also dem `imageFile`-Mitglied des `Item`-Objekts. Da das `imageFile`-Mitglied einen Zeichenfolgenwert enthält, können Sie auf den Dateinamen des Bildes über den Wert des untergeordneten Mitglieds `S` von `imageFile` zugreifen.

Durchblättern von zurückgegebenen Daten

Manchmal umfasst der Inhalt der von einer Service-Anforderung zurückgegebenen `data`-Eigenschaft mehrere Seiten. Sie können auf die nächste Datenseite mit dem Aufruf der `response.nextPage`-Methode zugreifen. Diese Methode sendet eine neue Anforderung. Die Antwort auf die Anforderung kann entweder mit einem Callback oder einem Listener für Erfolg oder Fehler erfasst werden.

Sie können prüfen, ob die von einer Service-Anforderung zurückgegebenen Daten zusätzliche Datenseiten haben, indem Sie die `response.hasNextPage`-Methode aufrufen. Diese Methode gibt einen booleschen Wert zurück, der angibt, ob der Aufruf von `response.nextPage` zusätzliche Daten zurückgibt.

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

Zugreifen auf Fehlerinformationen aus einem Antwortobjekt

Die `error`-Eigenschaft des `AWS.Response`-Objekts enthält die verfügbaren Fehlerdaten bei einem Service-Fehler oder einem Übertragungsfehler. Der zurückgegebene Fehler hat das folgende Format.

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

Im Fall eines Fehlers ist der Wert der `data`-Eigenschaft gleich `null`. Wenn Sie Ereignisse verarbeiten, die sich in einem Fehlerzustand befinden können, überprüfen Sie immer, ob die `error`-Eigenschaft festgelegt wurde, bevor Sie versuchen, auf den Wert der `data`-Eigenschaft zuzugreifen.

Zugreifen auf das ursprüngliche Anforderungsobjekt

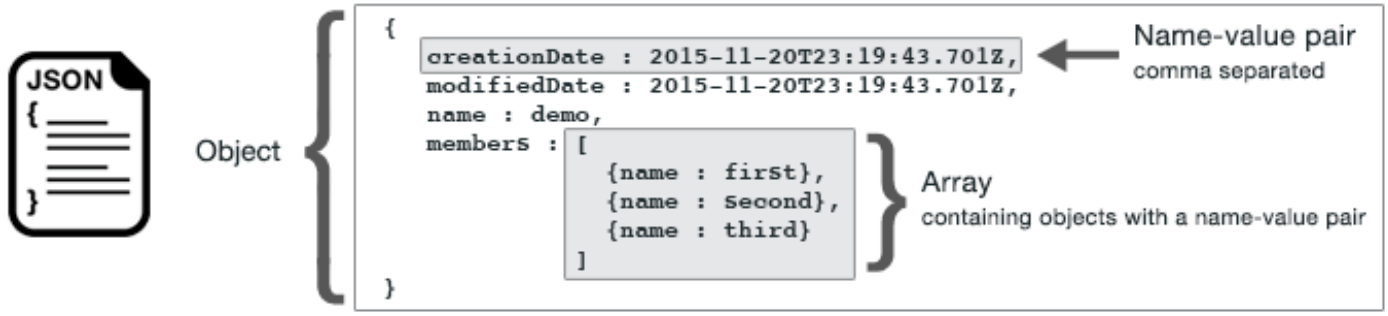
Die `request`-Eigenschaft bietet Zugriff auf das ursprüngliche `AWS.Request`-Objekt. Es kann nützlich sein, Bezug auf das ursprüngliche `AWS.Request`-Objekt zu nehmen, um auf die ursprünglichen Parameter zuzugreifen, die es gesendet hat. Im folgenden Beispiel wird die `request`-Eigenschaft verwendet, um auf den `Key`-Parameter der ursprünglichen Service-Anforderung zuzugreifen.

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

Arbeiten mit JSON

JSON ist sowohl visuell lesbares als auch ein maschinenlesbares Format für den Datenaustausch. Der Name JSON ist zwar eine Abkürzung für JavaScript Object Notation, das Format von JSON ist jedoch unabhängig von jeder Programmiersprache.

Das SDK für JavaScript verwendet JSON, um Daten an Serviceobjekte zu senden, wenn Anfragen gestellt werden, und empfängt Daten von Serviceobjekten als JSON. Weitere Informationen über JSON finden Sie auf der Website json.org.



JSON stellt Daten auf zwei Arten dar:

- Ein Objekt, bei dem es sich um eine ungeordnete Sammlung von Name-Wert-Paaren handelt. Ein Objekt wird innerhalb von zwei Klammern ({ und }) definiert. Jedes Name-Wert-Paar beginnt mit dem Namen, gefolgt von einem Doppelpunkt und dem Wert. Name-Wert-Paare werden durch Kommas voneinander getrennt.

- Ein Array, bei dem es sich um eine geordnete Sammlung von Werten handelt. Ein Array wird innerhalb von zwei Klammern (`[` und `]`) definiert. Elemente im Array werden durch Kommas voneinander getrennt.

Hier sehen Sie ein Beispiel für ein JSON-Objekt mit einem Array von Objekten, in dem die Objekte Karten in einem Kartenspiel darstellen. Jede Karte wird durch zwei Namen-Wert-Paare definiert. Eines davon gibt einen eindeutigen Wert an, um die Karte zu identifizieren, und das andere gibt die URL an, die auf das Kartenbild verweist.

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

JSON als Service-Objektparameter

Hier ist ein Beispiel für einfaches JSON, das verwendet wird, um die Parameter eines Aufrufs eines Lambda-Serviceobjekts zu definieren.

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

Das `pullParams`-Objekt wird durch drei Name-Wert-Paare definiert, die durch Kommas innerhalb einer linken und einer rechten Klammer getrennt sind. Beim Bereitstellen von Parametern für einen Service-Objektmethodenaufruf werden die Namen durch die Parameternamen für die Service-Objektmethode bestimmt, die Sie aufrufen möchten. Beim Aufrufen einer Lambda-Funktion sind,, und die Parameter `FunctionNameInvocationType`, die verwendet `LogType` werden, um die `invoke` Methode für ein Lambda-Serviceobjekt aufzurufen.

Wenn Sie Parameter an einen Methodenaufruf eines Serviceobjekts übergeben, stellen Sie das JSON-Objekt für den Methodenaufruf bereit, wie im folgenden Beispiel für den Aufruf einer Lambda-Funktion gezeigt.

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

Zurückgeben von Daten als JSON

JSON bietet eine standardisierte Methode, um Daten zwischen Teilen eine Anwendung zu übermitteln, die mehrere Werte gleichzeitig senden müssen. Die Methoden von Client-Klassen in der API geben gewöhnlich JSON im `data`-Parameter zurück, der an ihre Callback-Funktionen übergeben wird. Hier ist zum Beispiel ein Aufruf der `getBucketCors` Methode der Amazon S3 S3-Clientklasse.

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

Der Wert von `data` ist ein JSON-Objekt, in diesem Beispiel JSON, das die aktuelle CORS-Konfiguration für einen angegebenen Amazon S3 S3-Bucket beschreibt.

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

```
    ]
}
```

SDK für JavaScript Codebeispiele

Die Themen in diesem Abschnitt enthalten Beispiele für die Verwendung von AWS SDK for JavaScript mit den APIs verschiedener Services zum Ausführen allgemeiner Aufgaben.

Den Quellcode für diese und andere Beispiele finden Sie im Repository für AWS Dokumentationscodebeispiele unter GitHub. Um ein neues Codebeispiel für das AWS-Dokumentationsteam vorzuschlagen, dessen Erstellung es erwägen soll, erstellen Sie eine neue Anfrage. Das Team möchte Codebeispiele erstellen, die breitere Szenarien und Anwendungsfälle abdecken, im Vergleich zu einfachen Codeausschnitten, die nur einzelne API-Aufrufe abdecken. Eine Anleitung dazu finden Sie im Abschnitt Autorencode in den Richtlinien für Beiträge.

Themen

CloudWatch Amazon-Beispiele

Amazon CloudWatch (CloudWatch) ist ein Webservice, der Ihre Amazon Web Services Services-Ressourcen und -Anwendungen, auf denen Sie laufen, AWS in Echtzeit überwacht. Sie können CloudWatch damit Metriken sammeln und verfolgen. Dabei handelt es sich um Variablen, die Sie für Ihre Ressourcen und Anwendungen messen können. CloudWatch Alarme senden Benachrichtigungen oder nehmen auf der Grundlage von von Ihnen festgelegter Regeln automatisch Änderungen an den Ressourcen vor, die Sie überwachen.

Die JavaScript API für CloudWatch wird über die `AWS.CloudWatchLogs` Client-Klassen `AWS.CloudWatchAWS.CloudWatchEvents`, und verfügbar gemacht. Weitere Informationen zur Verwendung der CloudWatch Client-Klassen finden Sie unter `Class: AWS.CloudWatchClass: AWS.CloudWatchEvents`, und `Class: AWS.CloudWatchLogs`in der API-Referenz.

Themen

- Alarme in Amazon erstellen CloudWatch
- Alarmaktionen in Amazon verwenden CloudWatch
- Metriken von Amazon abrufen CloudWatch
- Ereignisse an Amazon CloudWatch Events senden
- Abonnementfilter in Amazon CloudWatch Logs verwenden

Alarme in Amazon erstellen CloudWatch

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie grundlegende Informationen zu Ihren CloudWatch Alarmen ab.
- So erstellen und löschen Sie einen CloudWatch Alarm.

Das Szenario

Ein Alarm überwacht eine Metrik über einen bestimmten, von Ihnen definierten Zeitraum und führt eine oder mehrere Aktionen durch, die vom Wert der Metrik im Vergleich zu einem festgelegten Schwellenwert in einer Reihe von Zeiträumen abhängt.

In diesem Beispiel werden eine Reihe von Node.js-Modulen verwendet, um Alarme in CloudWatch zu erstellen. Die Module von Node.js verwenden das SDK JavaScript , um Alarme mithilfe der folgenden Methoden der `AWS.CloudWatch` Client-Klasse zu erstellen:

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

Weitere Informationen zu CloudWatch Alarmen finden Sie unter CloudWatch Amazon-Alarme erstellen im CloudWatch Amazon-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Beschreiben von Alarmen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cw_describealarms.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um darauf zuzugreifen CloudWatch, erstellen Sie ein `AWS.CloudWatch` Serviceobjekt. Erstellen Sie ein JSON-Objekt, um die Parameter für das Abrufen von Alarmbeschreibungen zu speichern und begrenzen Sie dabei die zurückgegebenen Alarme auf diejenigen mit Status `INSUFFICIENT_DATA`. Rufen Sie dann die `describeAlarms`-Methode des `AWS.CloudWatch`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cw_describealarms.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Einen Alarm für eine CloudWatch Metrik erstellen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cw_putmetricalarm.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um darauf zuzugreifen CloudWatch, erstellen Sie ein `AWS.CloudWatch` Serviceobjekt. Erstellen Sie ein JSON-Objekt für die Parameter, die zum Erstellen eines Alarms auf der Grundlage einer Metrik erforderlich sind, in diesem Fall der CPU-Auslastung einer Amazon EC2 EC2-Instance. Die restlichen Parameter werden so festgelegt, dass der Alarm ausgelöst wird, wenn die Metrik einen Grenzwert von 70 Prozent überschreitet. Rufen Sie dann die `describeAlarms`-Methode des `AWS.CloudWatch`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cw_putmetricalarm.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen eines Alarms

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cw_deletealarms.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um darauf zuzugreifen CloudWatch, erstellen Sie ein `AWS.CloudWatch` Serviceobjekt. Erstellen Sie ein JSON-Objekt, um die Namen der Alarme zu speichern, die Sie löschen möchten. Rufen Sie dann die `deleteAlarms`-Methode des `AWS.CloudWatch`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cw_deletealarms.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Alarmaktionen in Amazon verwenden CloudWatch

Dieses Node.js-Codebeispiel zeigt:

- So ändern Sie den Status Ihrer Amazon EC2 EC2-Instances automatisch auf der Grundlage eines CloudWatch Alarms.

Das Szenario

Mithilfe von Alarmaktionen können Sie Alarme erstellen, die Ihre Amazon EC2 EC2-Instances automatisch stoppen, beenden, neu starten oder wiederherstellen. Sie können die Aktionen zum Anhalten oder Beenden nutzen, wenn eine Instance nicht mehr ausgeführt werden muss. Sie können die Aktionen zum Neustarten oder Wiederherstellen verwenden, um diese Instances automatisch neu zu starten.

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um eine Alarmaktion zu definieren CloudWatch , die den Neustart einer Amazon EC2 EC2-Instance auslöst. Die Module Node.js verwenden das SDK für JavaScript die Verwaltung von Amazon EC2 EC2-Instances mithilfe der folgenden Methoden der `CloudWatch` Client-Klasse:

- `enableAlarmActions`
- `disableAlarmActions`

Weitere Informationen zu CloudWatch Alarmaktionen finden Sie unter Erstellen von Alarmen zum Stoppen, Beenden, Neustarten oder Wiederherstellen einer Instance im CloudWatch Amazon-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine IAM-Rolle, deren Richtlinie die Erlaubnis erteilt, eine Amazon EC2 EC2-Instance zu beschreiben, neu zu starten, zu stoppen oder zu beenden. Weitere Informationen zum Erstellen einer IAM-Rolle finden Sie unter Creating a Role to Delegate Permissions to an AWS Service im IAM-Benutzerhandbuch.

Verwenden Sie die folgende Rollenrichtlinie beim Erstellen der IAM-Rolle.

```
{
```

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

Konfigurieren Sie das SDK für, JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt.

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

Erstellen und Aktivieren von Alarmaktionen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cw_enablealarmactions.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um darauf zuzugreifen CloudWatch, erstellen Sie ein `AWS.CloudWatch` Serviceobjekt.

Erstellen Sie ein JSON-Objekt für die Parameter, die für das Erstellen eines Alarms erforderlich sind. Geben Sie dabei `ActionsEnabled` als `true` und ein Array von ARNs für die Aktionen an, die der Alarm auslösen soll. Rufen Sie die `putMetricAlarm`-Methode des `AWS.CloudWatch`-Serviceobjekts auf, die den Alarm erstellt, wenn er nicht vorhanden ist, oder diesen aktualisiert, sofern vorhanden.

Erstellen Sie in der Callback-Funktion für nach erfolgreichem Abschluss ein JSON-Objekt`putMetricAlarm`, das den Namen des CloudWatch Alarms enthält. Rufen Sie die `enableAlarmActions`-Methode auf, um die Alarmaktion zu aktivieren.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cw_enablealarmactions.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Deaktivieren von Alarmaktionen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cw_disablealarmactions.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um darauf zuzugreifen CloudWatch, erstellen Sie ein `AWS.CloudWatch` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das den Namen des CloudWatch Alarms enthält. Rufen Sie die `disableAlarmActions`-Methode auf, um die Aktionen für diesen Alarm zu deaktivieren.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cw_disablealarmactions.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Metriken von Amazon abrufen CloudWatch

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie eine Liste veröffentlichter CloudWatch Metriken ab.
- So veröffentlichen Sie Datenpunkte in CloudWatch Metriken.

Das Szenario

Metriken sind Daten über die Leistung Ihrer Systeme. Sie können die detaillierte Überwachung einiger Ressourcen, wie z. B. Ihrer Amazon EC2 EC2-Instances, oder Ihrer eigenen Anwendungsmetriken aktivieren.

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Metriken von Amazon Events abzurufen CloudWatch und Ereignisse an Amazon CloudWatch Events zu senden. Die Module Node.js verwenden das SDK JavaScript , um Metriken CloudWatch mithilfe der folgenden Methoden der `CloudWatch` Client-Klasse abzurufen:

- `listMetrics`
- `putMetricData`

Weitere Informationen zu CloudWatch Metriken finden Sie unter Using Amazon CloudWatch Metrics im CloudWatch Amazon-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.

- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Auflisten von Metriken

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cw_listmetrics.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um darauf zuzugreifen CloudWatch, erstellen Sie ein `AWS.CloudWatch` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Auflisten von Metriken innerhalb des `AWS/Logs`-Namespace. Rufen Sie die `listMetrics`-Methode zum Auflisten der `IncomingLogEvents`-Metrik auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cw_listmetrics.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Übermitteln von benutzerdefinierten Metriken

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cw_putmetricdata.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um darauf zuzugreifen CloudWatch, erstellen Sie ein `AWS.CloudWatch` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Übermitteln eines Datenpunktes an die benutzerdefinierte `PAGES_VISITED`-Metrik. Rufen Sie die `putMetricData`-Methode auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cw_putmetricdata.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Ereignisse an Amazon CloudWatch Events senden

Dieses Node.js-Codebeispiel zeigt:

- So erstellen und aktualisieren Sie eine Regel für das Auslösen eines Ereignisses.
- So definieren Sie ein oder mehrere Ziele für die Reaktion auf ein Ereignis.
- So senden Sie Ereignisse, die Zielen zur Bearbeitung zugeordnet sind.

Das Szenario

CloudWatch Events liefert nahezu in Echtzeit einen Stream von Systemereignissen, die Änderungen an den Amazon Web Services Services-Ressourcen für verschiedene Ziele beschreiben. Mit einfachen Regeln können Sie Ereignisse zuordnen und sie zu einer oder mehreren Zielfunktionen oder Streams umleiten.

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Ereignisse an CloudWatch Events zu senden. Die Module Node.js verwenden das SDK für JavaScript die Verwaltung von Instanzen mithilfe der folgenden Methoden der `CloudWatchEvents` Client-Klasse:

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

Weitere Informationen zu CloudWatch Veranstaltungen finden Sie unter Hinzufügen von Ereignissen mit PutEvents im Amazon CloudWatch Events-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie mithilfe des Hello-World-Blueprints eine Lambda-Funktion, die als Ziel für Ereignisse dient. Wie das geht, erfahren Sie unter Schritt 1: AWS Lambda Funktion erstellen im Amazon CloudWatch Events-Benutzerhandbuch.
- Erstellen Sie eine IAM-Rolle, deren Richtlinie Berechtigungen für CloudWatch Veranstaltungen gewährt, und dazu gehört auch `events.amazonaws.com` die Rolle „Vertrauenswürdige Entität“. Weitere Informationen zum Erstellen einer IAM-Rolle finden Sie unter Creating a Role to Delegate Permissions to an AWS Service im IAM-Benutzerhandbuch.

Verwenden Sie die folgende Rollenrichtlinie beim Erstellen der IAM-Rolle.

```
{
   "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"
      }
   ]
```

```
}
```

Verwenden Sie beim Erstellen einer IAM-Rolle die folgende Vertrauensstellung.

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

Erstellen einer geplanten Regel

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cwe_putrule.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf CloudWatch Ereignisse zuzugreifen, erstellen Sie ein `AWS.CloudWatchEvents` Serviceobjekt. Erstellen Sie ein JSON-Objekt, welches die für die neue geplante Regel erforderlichen Parameter enthält, darunter die folgenden:

- Einen Namen für die Regel
- Der ARN der IAM-Rolle, die Sie zuvor erstellt haben
- Einen Ausdruck zum Auslösen der Regel alle fünf Minuten

Rufen Sie die `putRule`-Methode auf, um die Regel zu erstellen. Der Callback gibt den ARN der neuen oder aktualisierten Regel zurück.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cwe_putrule.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Ein AWS Lambda Funktionsziel hinzufügen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cwe_puttargets.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf CloudWatch Ereignisse zuzugreifen, erstellen Sie ein `AWS.CloudWatchEvents` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den Parametern, die zur Angabe der Regel erforderlich sind, an die Sie das Ziel anhängen möchten, einschließlich des ARN der von Ihnen erstellten Lambda-Funktion. Rufen Sie die `putTargets`-Methode des `AWS.CloudWatchEvents`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cwe_puttargets.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Senden von Ereignissen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cwe_putevents.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf CloudWatch Ereignisse zuzugreifen, erstellen Sie ein `AWS.CloudWatchEvents` Serviceobjekt. Erstellen Sie ein JSON-Objekt, welches die für das Senden von Ereignissen erforderlichen Parameter enthält. Geben Sie für jedes Ereignis die Quelle des Ereignisses, die ARNs aller vom Ereignis betroffenen Ressourcen sowie Details zum Ereignis an. Rufen Sie die `putEvents`-Methode des `AWS.CloudWatchEvents`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cwe_putevents.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Abonnementfilter in Amazon CloudWatch Logs verwenden

Dieses Node.js-Codebeispiel zeigt:

- So erstellen und löschen Sie Filter für Protokollereignisse in CloudWatch Logs.

Das Szenario

Abonnements bieten Zugriff auf einen Echtzeit-Feed mit Protokollereignissen aus CloudWatch Logs und leiten diesen Feed an andere Dienste weiter, z. B. an einen Amazon Kinesis Kinesis-Stream oder AWS Lambda zur benutzerdefinierten Verarbeitung, Analyse oder zum Laden in andere Systeme. Ein Abonnementfilter definiert das Muster, nach dem gefiltert wird, welche Protokollereignisse an Ihre AWS Ressource übermittelt werden.

In diesem Beispiel werden eine Reihe von Modulen von Node.js verwendet, um einen Abonnementfilter in CloudWatch Logs aufzulisten, zu erstellen und zu löschen. Das Ziel für

die Protokollereignisse ist eine Lambda-Funktion. Die Module von Node.js verwenden das SDK JavaScript zur Verwaltung von Abonnementfiltern mithilfe der folgenden Methoden der `CloudWatchLogs` Client-Klasse:

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

Weitere Informationen zu CloudWatch Logs-Abonnements finden Sie unter Echtzeitverarbeitung von Protokolldaten mit Abonnements im Amazon CloudWatch Logs-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine Lambda-Funktion als Ziel für Protokollereignisse. Dazu benötigen Sie den ARN dieser Funktion. Weitere Informationen zur Einrichtung einer Lambda-Funktion finden Sie unter Abonnementfilter mit AWS Lambda im Amazon CloudWatch Logs-Benutzerhandbuch.
- Erstellen Sie eine IAM-Rolle, deren Richtlinie die Berechtigung zum Aufrufen der von Ihnen erstellten Lambda-Funktion und vollen Zugriff auf CloudWatch Logs gewährt, oder wenden Sie die folgende Richtlinie auf die Ausführungsrolle an, die Sie für die Lambda-Funktion erstellen. Weitere Informationen zum Erstellen einer IAM-Rolle finden Sie unter Creating a Role to Delegate Permissions to an AWS Service im IAM-Benutzerhandbuch.

Verwenden Sie die folgende Rollenrichtlinie beim Erstellen der IAM-Rolle.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
```

```
         "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "lambda:InvokeFunction"
         ],
         "Resource": [
            "*"
         ]
      }
   ]
}
```

Beschreiben von bestehenden Abonnementfiltern

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cwl_describesubscriptionfilters.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf CloudWatch Protokolle zuzugreifen, erstellen Sie ein `AWS.CloudWatchLogs` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für die Beschreibung Ihrer bestehenden Filter, einschließlich der Namen der Protokollgruppe sowie der maximalen Anzahl der Filter, die Sie beschreiben möchten. Rufen Sie die `describeSubscriptionFilters`-Methode auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cwl_describesubscriptionfilters.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Erstellen eines Abonnementfilters

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cwl_putsubscriptionfilter.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf CloudWatch Logs zuzugreifen, erstellen Sie ein `AWS.CloudWatchLogs` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die Parameter enthält, die zum Erstellen eines Filters erforderlich sind, einschließlich des ARN der Ziel-Lambda-Funktion, des Namens des Filters, des Zeichenkettenmusters für die Filterung und des Namens der Protokollgruppe. Rufen Sie die `putSubscriptionFilters`-Methode auf.

```
// 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);
```

```
  }
});
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cwl_putsubscriptionfilter.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen eines Abonnementfilters

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `cwl_deletesubscriptionfilters.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf CloudWatch Logs zuzugreifen, erstellen Sie ein `AWS.CloudWatchLogs` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Löschen eines Filters, einschließlich des Filternamens und des Namens der Protokollgruppe. Rufen Sie die `deleteSubscriptionFilters`-Methode auf.

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

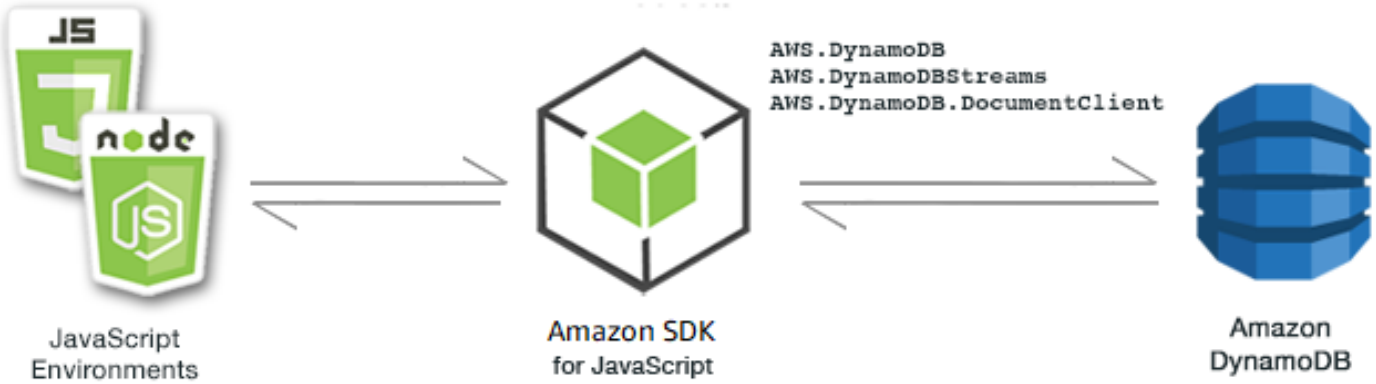
Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node cwl_deletesubscriptionfilter.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Amazon DynamoDB-Beispiele

Amazon DynamoDB ist eine vollständig verwaltete NoSQL-Cloud-Datenbank, die sowohl Dokument- als auch Key-Value-Speichermodelle unterstützt. Sie erstellen schemalose Tabellen für Daten – ohne Bereitstellung oder Wartung dedizierter Datenbankserver.



Die JavaScript API für DynamoDB wird über die `AWS.DynamoDB.DocumentClient` Clientklassen `AWS.DynamoDB`AWS.DynamoDBStreams, und verfügbar gemacht. Weitere Informationen zur Verwendung der DynamoDB-Clientklassen finden Sie unter `Class: AWS.DynamoDB`Class: `AWS.DynamoDBStreams`, und `Class: AWS.DynamoDB.DocumentClient`in der API-Referenz.

Themen

- Tabellen in DynamoDB erstellen und verwenden
- Lesen und Schreiben eines einzelnen Elements in DynamoDB
- Batch-Elemente in DynamoDB lesen und schreiben
- Abfragen und Scannen einer DynamoDB-Tabelle
- Verwenden des DynamoDB-Dokumentenclients

Tabellen in DynamoDB erstellen und verwenden

Dieses Node.js-Codebeispiel zeigt:

- So erstellen und verwalten Sie Tabellen, die zum Speichern und Abrufen von Daten aus DynamoDB verwendet werden.

Das Szenario

Ebenso wie andere Datenbanksysteme speichert auch DynamoDB Daten in Tabellen. Eine DynamoDB-Tabelle ist eine Sammlung von Daten, die in Elementen organisiert sind, die Zeilen entsprechen. Um Daten in DynamoDB zu speichern oder darauf zuzugreifen, erstellen Sie Tabellen und arbeiten mit ihnen.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um grundlegende Operationen mit einer DynamoDB-Tabelle durchzuführen. Der Code verwendet das SDK JavaScript , um Tabellen mithilfe der folgenden Methoden der `AWS.DynamoDB` Client-Klasse zu erstellen und mit ihnen zu arbeiten:

- createTable
- listTables
- describeTable
- deleteTable

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Erstellen einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_createtable.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen

Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Erstellen einer Tabelle, das in diesem Beispiel den Namen und Datentyp für jedes Attribut, das Schlüsselschema, den Namen der Tabelle und die Durchsatzeinheiten beinhaltet, die bereitgestellt werden sollen. Rufen Sie die `createTable` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_createtable.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Auflisten von Tabellen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_listtables.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Auflisten Ihrer Tabellen, das in diesem Beispiel die Anzahl der aufgeführten Tabellen auf 10 begrenzt. Rufen Sie die `listTables` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_listtables.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Beschreiben einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_describetable.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für die Beschreibung einer Tabelle, das in diesem Beispiel den Namen der Tabelle beinhaltet, der als Befehlszeilenparameter bereitgestellt wird. Rufen Sie die `describeTable` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_describetable.js TABLE_NAME
```

Diesen Beispielcode finden Sie hier auf. GitHub

Löschen einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_deletetable.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern zum Löschen einer Tabelle, das in diesem Beispiel den Namen der Tabelle beinhaltet, der als Befehlszeilenparameter bereitgestellt wird. Rufen Sie die `deleteTable` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_deletetable.js TABLE_NAME
```

Diesen Beispielcode finden Sie hier auf. GitHub

Lesen und Schreiben eines einzelnen Elements in DynamoDB

Dieses Node.js-Codebeispiel zeigt:

- So fügen Sie ein Element zu einer DynamoDB-Tabelle hinzu.
- So rufen Sie ein Element in einer DynamoDB-Tabelle ab.
- So löschen Sie ein Element in einer DynamoDB-Tabelle.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um ein Element in einer DynamoDB-Tabelle zu lesen und zu schreiben, indem Sie die folgenden Methoden der `AWS.DynamoDB` Client-Klasse verwenden:

- putItem
- getItem
- deleteItem

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine DynamoDB-Tabelle, auf deren Elemente Sie zugreifen können. Weitere Hinweise zum Erstellen einer DynamoDB-Tabelle finden Sie unter. Tabellen in DynamoDB erstellen und verwenden

Schreiben eines Elements

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_putitem.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Hinzufügen eines Elements, das in diesem Beispiel den Namen der Tabelle und eine Map beinhaltet, die die festzulegenden Attribute und Werte für jedes Attribut definiert. Rufen Sie die `putItem` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_putitem.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Abrufen eines Elements

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_getitem.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Zum Identifizieren des abzurufenden Elements müssen Sie den Wert des Primärschlüssels für dieses Element in der Tabelle angeben. Die `getItem`-Methode gibt standardmäßig alle Attributwerte zurück, die für das Element definiert sind. Um nur eine Teilmenge aller möglichen Attributwerte abzurufen, geben Sie einen Projektionsausdruck an.

Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abrufen eines Elements, das in diesem Beispiel den Namen der Tabelle, den Namen und Wert des Schlüssels für das abzurufende Element und einen Projektionsausdruck beinhaltet, der das abzurufende Elementattribut identifiziert. Rufen Sie die `getItem` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_getitem.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Löschen eines Elements

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_deleteitem.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern zum Löschen eines Elements, das in diesem Beispiel den Namen der Tabelle und den Schlüsselnamen sowie den Wert für das zu löschende Element beinhaltet. Rufen Sie die `deleteItem` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_deleteitem.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Batch-Elemente in DynamoDB lesen und schreiben

Dieses Node.js-Codebeispiel zeigt:

- So lesen und schreiben Sie Stapel von Elementen in einer DynamoDB-Tabelle.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um einen Stapel von Elementen in eine DynamoDB-Tabelle einzufügen und einen Stapel von Elementen zu lesen. Der Code verwendet das SDK für JavaScript die Ausführung von Batch-Lese- und Schreiboperationen mit den folgenden Methoden der DynamoDB-Clientklasse:

- batchGetItem
- batchWriteItem

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine DynamoDB-Tabelle, auf deren Elemente Sie zugreifen können. Weitere Hinweise zum Erstellen einer DynamoDB-Tabelle finden Sie unter. Tabellen in DynamoDB erstellen und verwenden

Lesen von Elementen im Stapel

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_batchgetitem.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abrufen eines Stapels von Elementen, das in diesem Beispiel den Namen einer oder mehrerer zu lesender Tabellen, die in jeder Tabelle zu lesenden Werte von Schlüsseln und den Projektionsausdruck beinhaltet, der die zurückzugebenden Attribute angibt. Rufen Sie die `batchGetItem` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_batchgetitem.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Schreiben von Elementen im Stapel

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_batchwriteitem.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abrufen eines Stapels von Elementen, das in diesem Beispiel die Tabelle, in die Sie Elemente schreiben möchten, den/die Schlüssel, den/die Sie für jedes Element schreiben möchten, und die Attribute zusammen mit ihren Werten beinhaltet. Rufen Sie die `batchWriteItem` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_batchwriteitem.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Abfragen und Scannen einer DynamoDB-Tabelle

Dieses Node.js-Codebeispiel zeigt:

- So fragen Sie eine DynamoDB-Tabelle ab und scannen sie nach Elementen.

Das Szenario

Die Abfrage findet Elemente in einer Tabelle oder einem Sekundärindex nur über Primärschlüsselattributwerte. Sie müssen einen Partitionsschlüsselnamen und einen Wert angeben, nach dem gesucht werden soll. Sie können auch einen Sortierschlüsselnamen und Wert angeben, und einen Vergleichsoperator verwenden, um die Suchergebnisse zu verfeinern. Das Scannen findet Elemente, indem jedes Element in der angegebenen Tabelle überprüft wird.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um ein oder mehrere Elemente zu identifizieren, die Sie aus einer DynamoDB-Tabelle abrufen möchten. Der Code verwendet das

SDK JavaScript zum Abfragen und Scannen von Tabellen mithilfe der folgenden Methoden der DynamoDB-Clientklasse:

- query
- scan

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine DynamoDB-Tabelle, auf deren Elemente Sie zugreifen können. Weitere Hinweise zum Erstellen einer DynamoDB-Tabelle finden Sie unter. Tabellen in DynamoDB erstellen und verwenden

Abfragen einer Tabelle

In diesem Beispiel wird eine Tabelle abgefragt, die Informationen zu Episoden einer Videoserie enthält, und die Episodentitel und -untertitel von Episoden der zweiten Staffel nach Episode 9 zurückgegeben, die einen angegebene Begriff im Untertitel enthalten.

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_query.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abfragen der Tabelle, das in diesem Beispiel den Tabellennamen, die von der Abfrage benötigten `ExpressionAttributeValues`, einen `KeyConditionExpression`, der diese Werte verwendet, um zu definieren, welche Elemente die Abfrage zurück gibt, und die Namen der für jedes Element zurückzugebenden Attributwerte beinhaltet. Rufen Sie die `query` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_query.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Scannen einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddb_scan.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Scannen der Tabelle nach Elementen, das in diesem Beispiel den Namen der Tabelle, die Liste der Attributwerte, die für jedes übereinstimmende Element zurückgegeben werden sollen, und einen Ausdruck zum Filtern des Ergebnissatzes beinhaltet, um nach Elementen mit einem bestimmten Begriff zu suchen. Rufen Sie die `scan` Methode des DynamoDB-Dienstobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddb_scan.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Verwenden des DynamoDB-Dokumentenclients

Dieses Node.js-Codebeispiel zeigt:

- So greifen Sie mit dem Document Client auf eine DynamoDB-Tabelle zu.

Das Szenario

Der DynamoDB-Dokumentclient vereinfacht die Arbeit mit Elementen, indem er den Begriff der Attributwerte abstrahiert. Diese Abstraktion annotiert native JavaScript Typen, die als Eingabeparameter bereitgestellt werden, und konvertiert annotierte Antwortdaten in native Typen. JavaScript

Weitere Informationen zur DynamoDB Document Client-Klasse finden Sie `AWS.DynamoDB.DocumentClient`in der API-Referenz. Weitere Informationen zur Programmierung mit Amazon DynamoDB finden Sie unter Programmieren mit DynamoDB im Amazon DynamoDB Developer Guide.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um mithilfe des Document Client grundlegende Operationen an einer DynamoDB-Tabelle durchzuführen. Der Code verwendet das SDK JavaScript zum Abfragen und Scannen von Tabellen mithilfe der folgenden Methoden der DynamoDB Document Client-Klasse:

- get
- put
- update
- query
- delete

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine DynamoDB-Tabelle, auf deren Elemente Sie zugreifen können. Weitere Hinweise zum Erstellen einer DynamoDB-Tabelle mit dem SDK für JavaScript finden Sie unter. Tabellen in DynamoDB erstellen und verwenden Sie können auch die DynamoDB-Konsole verwenden, um eine Tabelle zu erstellen.

Abrufen eines Elements aus einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddbdoc_get.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB.DocumentClient` Objekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abrufen eines Elements aus der Tabelle, das in diesem Beispiel den Namen der Tabelle, den Namen des Hash-Schlüssels in dieser Tabelle und den Wert des Hash-Schlüssels für das abzurufende Element beinhaltet. Rufen Sie die `get` Methode des DynamoDB-Dokumentclients auf.

```
// 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);
  }
```

```
});
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddbdoc_get.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Einfügen eines Elements in eine Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddbdoc_put.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB.DocumentClient` Objekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Schreiben eines Elements in die Tabelle. Dieses beinhaltet in diesem Beispiel den Namen der Tabelle und eine Beschreibung des hinzuzufügenden oder zu aktualisierenden Elements, das den Hashkey und Wert sowie die Namen und Werte für Attribute beinhaltet, die für das Element festgelegt werden sollen. Rufen Sie die `put` Methode des DynamoDB-Dokumentclients auf.

```
// 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);
  }
```

```
});
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddbdoc_put.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Aktualisieren eines Elements in einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddbdoc_update.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB.DocumentClient` Objekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Schreiben eines Elements in die Tabelle, das in diesem Beispiel den Namen der Tabelle, den Schlüssel des zu aktualisierenden Elements, eine Reihe von `UpdateExpressions` beinhaltet, die die Attribute des Elements definieren, das mit Token aktualisiert werden soll, denen Sie Werte in den `ExpressionAttributeValues`-Parametern zuweisen. Rufen Sie die `update` Methode des DynamoDB-Dokumentclients auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddbdoc_update.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Abfragen einer Tabelle

In diesem Beispiel wird eine Tabelle abgefragt, die Informationen zu Episoden einer Videoserie enthält, und die Episodentitel und -untertitel von Episoden der zweiten Staffel nach Episode 9 zurückgegeben, die einen angegebene Begriff im Untertitel enthalten.

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddbdoc_query.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB.DocumentClient` Objekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abfragen der Tabelle, das in diesem Beispiel den Tabellennamen, die von der Abfrage benötigten `ExpressionAttributeValues` und einen `KeyConditionExpression` beinhaltet, der diese Werte verwendet, um zu definieren, welche Elemente die Abfrage zurück gibt. Rufen Sie die `query` Methode des DynamoDB-Dokumentclients auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddbdoc_query.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Löschen eines Elements aus einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ddbdoc_delete.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf DynamoDB zuzugreifen, erstellen Sie ein `AWS.DynamoDB.DocumentClient` Objekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Löschen eines Elements in der Tabelle, das in diesem Beispiel den Namen der Tabelle sowie den Namen und Wert des Hashkey des zu löschenden Elements beinhaltet. Rufen Sie die `delete` Methode des DynamoDB-Dokumentclients auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ddbdoc_delete.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Amazon EC2-Beispiele

Amazon Elastic Compute Cloud (Amazon EC2) ist ein Webservice, der virtuelles Server-Hosting in der Cloud bereitstellt. Er wurde entwickelt, um webweites Cloud-Computing durch Bereitstellung skalierbarer Rechenkapazität für Entwickler einfacher zu machen.

Die JavaScript API für Amazon EC2 wird über die `AWS.EC2` Client-Klasse verfügbar gemacht. Weitere Informationen zur Verwendung der Amazon EC2 EC2-Client-Klasse finden Sie `Class: AWS.EC2`in der API-Referenz.

Themen

Eine Amazon EC2 EC2-Instance erstellen

Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie eine Amazon EC2 EC2-Instance aus einem öffentlichen Amazon Machine Image (AMI).
- So erstellen Sie Tags und weisen sie der neuen Amazon EC2 EC2-Instance zu.

Informationen zum Beispiel

In diesem Beispiel verwenden Sie ein Modul Node.js, um eine Amazon EC2 EC2-Instance zu erstellen und ihr sowohl ein key pair als auch Tags zuzuweisen. Der Code verwendet das SDK JavaScript , um eine Instance mithilfe der folgenden Methoden der Amazon EC2 EC2-Client-Klasse zu erstellen und zu kennzeichnen:

- `runInstances`
- `createTags`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab.

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.

- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen eines Schlüsselpaares Details hierzu finden Sie unter Arbeiten mit Amazon EC2-Schlüsselpaaren. Sie verwenden den Namen des Schlüsselpaars in diesem Beispiel.

Erstellen und Markieren einer Instance

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_createinstances.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren.

Erstellen Sie ein Objekt, um die Parameter für die `runInstances`-Methode der AWS.EC2-Client-Klasse zu übergeben, einschließlich dem Namen des zuzuweisenden Schlüsselpaars und der ID des auszuführenden AMIs. Um die `runInstances`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon EC2-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die response im Promise-Callback.

Der nächste Code fügt einer neuen Instance ein `Name` Tag hinzu, das von der Amazon EC2 EC2-Konsole erkannt und im Feld Name der Instance-Liste angezeigt wird. Sie können einer Instance bis zu 50 Tags hinzufügen, die alle in einem einzigen Aufruf der `createTags`-Methode hinzugefügt werden können.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_createinstances.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verwalten von Amazon EC2 Instances

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie grundlegende Informationen zu Ihren Amazon EC2 EC2-Instances ab.
- So starten und beenden Sie die detaillierte Überwachung einer Amazon EC2 EC2-Instance.
- So starten und beenden Sie eine Amazon EC2 EC2-Instance.
- So starten Sie eine Amazon EC2 EC2-Instance neu.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js-Modulen zum Ausführen mehrerer einfacher Instance-Verwaltungsoperationen. Die Module Node.js verwenden das SDK für JavaScript die Verwaltung von Instances mithilfe dieser Amazon EC2 EC2-Client-Klassenmethoden:

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

Weitere Informationen zum Lebenszyklus von Amazon EC2 EC2-Instances finden Sie unter Instance-Lebenszyklus im Amazon EC2 EC2-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.

- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine Amazon EC2 EC2-Instance. Weitere Informationen zur Erstellung von Amazon EC2 EC2-Instances finden Sie unter Amazon EC2 EC2-Instances im Amazon EC2 EC2-Benutzerhandbuch oder Amazon EC2 EC2-Instances im Amazon EC2 EC2-Benutzerhandbuch.

Beschreiben Ihrer Instances

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_describeinstances.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Rufen Sie die `describeInstances` Methode des Amazon EC2-Serviceobjekts auf, um eine detaillierte Beschreibung Ihrer Instances abzurufen.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_describeinstances.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Verwalten der Instance-Überwachung

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_monitorinstances.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Fügen Sie die Instance-IDs der Instances hinzu, für die Sie die Überwachung steuern möchten.

Rufen Sie auf der Grundlage des Werts eines Befehlszeilenarguments (`0N` oder `0FF`) entweder die `monitorInstances` Methode des Amazon EC2-Serviceobjekts auf, um mit der detaillierten Überwachung der angegebenen Instances zu beginnen, oder rufen Sie die Methode auf. `unmonitorInstances` Verwenden Sie den `DryRun`-Parameter, um zu testen, ob Sie über die Berechtigung zum Ändern der Instance-Überwachung verfügen, bevor Sie versuchen, die Überwachung dieser Instances zu ändern.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein und legen Sie `ON` zum Starten der detaillierten Überwachung oder `OFF` zum Stoppen der Überwachung fest.

```
node ec2_monitorinstances.js ON
```

Diesen Beispielcode finden Sie hier unter. GitHub

Starten und Stoppen der Instances

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_startstopinstances.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Fügen Sie die Instance-IDs der Instances hinzu, die Sie starten oder stoppen möchten.

Rufen Sie basierend auf dem Wert eines Befehlszeilenarguments (`STARToderSTOP`) entweder die `startInstances` Methode des Amazon EC2-Serviceobjekts auf, um die angegebenen Instances zu starten, oder die `stopInstances` Methode, um sie zu stoppen. Verwenden Sie den `DryRun`-Parameter, um zu testen, ob Sie berechtigt sind, bevor Sie versuchen, die ausgewählten Instances zu starten oder zu stoppen.

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

```
}
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein und legen Sie `START` fest, um die Instances zu starten, oder `STOP`, um sie zu stoppen.

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

Diesen Beispielcode finden Sie hier unter. GitHub

Neustarten von Instances

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_rebootinstances.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein Amazon EC2-Serviceobjekt. Fügen Sie die Instance-IDs der Instances hinzu, die Sie neu starten möchten. Rufen Sie die `rebootInstances`-Methode des `AWS.EC2`-Serviceobjekts auf, um die angegebenen Instances neu zu starten. Verwenden Sie den `DryRun`-Parameter, um zu testen, ob Sie berechtigt sind, diese Instances neu zu starten, bevor Sie versuchen, sie neu zu starten.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_rebootinstances.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Arbeiten mit Amazon EC2-Schlüsselpaaren

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie grundlegende Informationen über Ihre Schlüsselpaare ab.
- So erstellen Sie ein key pair für den Zugriff auf eine Amazon EC2 EC2-Instance.
- So löschen Sie ein vorhandenes Schlüsselpaar.

Das Szenario

Amazon EC2 verwendet Kryptografie für öffentliche Schlüssel, um Anmeldeinformationen zu ver- und entschlüsseln. Bei der Kryptografie für öffentliche Schlüssel werden öffentliche Schlüssel eingesetzt, um Daten zu verschlüsseln. Der Empfänger entschlüsselt diese Daten dann mit einem privaten Schlüssel. Der öffentliche und der private Schlüssel werden als Schlüsselpaar bezeichnet.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um mehrere Amazon EC2 EC2-Schlüsselpaarverwaltungsvorgänge durchzuführen. Die Module Node.js verwenden das SDK für JavaScript die Verwaltung von Instances mithilfe der folgenden Methoden der Amazon EC2 EC2-Client-Klasse:

- `createKeyPair`

- `deleteKeyPair`
- `describeKeyPairs`

Weitere Informationen zu den Amazon EC2 EC2-Schlüsselpaaren finden Sie unter Amazon EC2 EC2-Schlüsselpaare im Amazon EC2 EC2-Benutzerhandbuch oder Amazon EC2 EC2-Schlüsselpaare und Windows-Instances im Amazon EC2 EC2-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Beschreiben Ihrer Schlüsselpaare

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_describekeypairs.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie ein leeres JSON-Objekt zum Speichern der Parameter, die von der `describeKeyPairs`-Methode zur Rückgabe von Beschreibungen für alle Ihre Schlüsselpaare benötigt werden. Sie können auch ein Array von Namen von Schlüsselpaaren im `KeyName`-Teil der Parameter der JSON-Datei für die `describeKeyPairs`-Methode bereitstellen.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_describekeypairs.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Erstellen eines Schlüsselpaars

Jedes Schlüsselpaar benötigt einen Namen. Amazon EC2 ordnet den öffentlichen Schlüssel dem Namen zu, den Sie als Schlüsselnamen angeben. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_createkeypair.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie die JSON-Parameter, um den Namen des Schlüsselpaars anzugeben, und übergeben sie dann, um die `createKeyPair` -Methode aufzurufen.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_createkeypair.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen eines Schlüsselpaars

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_deletekeypair.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie die JSON-Parameter, um den Namen des Schlüsselpaars anzugeben, das Sie löschen möchten. Rufen Sie anschließend die Methode `deleteKeyPair` auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_deletekeypair.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verwenden von Regionen und Availability Zones mit Amazon EC2

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie Beschreibungen für Regionen und Availability Zones ab.

Das Szenario

Amazon EC2 wird an mehreren Standorten weltweit gehostet. Diese Standorte bestehen aus - Regionen und Availability Zones. Jede -Region ist ein separater geografischer Bereich. Jede Region verfügt über mehrere isolierte Standorte, die als Availability Zones bezeichnet werden. Amazon EC2 bietet die Möglichkeit, Instances und Daten an mehreren Standorten zu platzieren.

In diesem Beispiel verwenden Sie eine Reihe von Node.js-Modulen zum Abrufen von Details über Regionen und Availability Zones. Die Module Node.js verwenden das SDK für JavaScript die Verwaltung von Instances mithilfe der folgenden Methoden der Amazon EC2 EC2-Client-Klasse:

- `describeAvailabilityZones`
- `describeRegions`

Weitere Informationen zu Regionen und Availability Zones finden Sie unter Regionen und Availability Zones im Amazon EC2 EC2-Benutzerhandbuch oder Regionen und Availability Zones im Amazon EC2 EC2-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Beschreiben von Regionen und Availability Zones

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_describeregionsandzones.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie ein leeres JSON-Objekt, das als Parameter übergeben werden soll, das alle verfügbaren Beschreibungen zurück gibt. Rufen Sie anschließend die Methoden `describeRegions` und `describeAvailabilityZones` auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_describeregionsandzones.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Arbeiten mit Sicherheitsgruppen in Amazon EC2

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie grundlegende Informationen über Ihre Sicherheitsgruppen ab.
- So erstellen Sie eine Sicherheitsgruppe für den Zugriff auf eine Amazon EC2 EC2-Instance.
- So löschen Sie eine bestehende Sicherheitsgruppe.

Das Szenario

Eine Amazon EC2-Sicherheitsgruppe fungiert als virtuelle Firewall, die den Datenverkehr für eine oder mehrere Instances steuert. Sie fügen jeder Sicherheitsgruppe Regeln hinzu, um den Datenaustausch mit den verknüpften Instances zu gestatten. Diese Regeln können Sie jederzeit ändern, wobei die neuen Regeln automatisch auf alle Instances der Sicherheitsgruppe angewendet werden.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um mehrere Amazon EC2 EC2-Operationen mit Sicherheitsgruppen durchzuführen. Die Module Node.js verwenden das SDK für JavaScript die Verwaltung von Instances mithilfe der folgenden Methoden der Amazon EC2 EC2-Client-Klasse:

- `describeSecurityGroups`
- `authorizeSecurityGroupIngress`
- `createSecurityGroup`
- `describeVpcs`
- `deleteSecurityGroup`

Weitere Informationen zu den Amazon EC2-Sicherheitsgruppen finden Sie unter Amazon EC2 Amazon Security Groups for Linux Instances im Amazon EC2-Benutzerhandbuch oder Amazon EC2-Sicherheitsgruppen für Windows-Instances im Amazon EC2 EC2-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Beschreiben Ihrer Sicherheitsgruppen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_describesecuritygroups.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das als Parameter übergeben werden soll, einschließlich der Gruppen-IDs für die Sicherheitsgruppen, die Sie beschreiben möchten. Rufen Sie dann die `describeSecurityGroups` Methode des Amazon EC2-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_describesecuritygroups.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Erstellen von Regeln und einer Sicherheitsgruppe

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_createsecuritygroup.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie ein JSON-Objekt für die Parameter, die den Namen der Sicherheitsgruppe, eine Beschreibung und die ID für die VPC festlegen. Übergeben Sie die Parameter an die `createSecurityGroup`-Methode.

Nachdem Sie die Sicherheitsgruppe erfolgreich erstellt haben, können Sie Regeln zum Zulassen des eingehenden Datenverkehrs definieren. Erstellen Sie ein JSON-Objekt für Parameter, die das IP-Protokoll und die eingehenden Ports angeben, auf denen die Amazon EC2 EC2-Instance Datenverkehr empfängt. Übergeben Sie die Parameter an die `authorizeSecurityGroupIngress`-Methode.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_createsecuritygroup.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Löschen einer Sicherheitsgruppe

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_deletesecuritygroup.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie die JSON-Parameter, um den Namen der Sicherheitsgruppe anzugeben, die Sie löschen möchten. Rufen Sie anschließend die Methode `deleteSecurityGroup` auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_deletesecuritygroup.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verwenden von Elastic IP-Adressen in Amazon EC2

Dieses Node.js-Codebeispiel zeigt:

- So fragen Sie Beschreibungen Ihrer Elastic IP-Adressen ab.
- So ordnen Sie eine Elastic IP-Adresse zu und geben sie frei.
- So verknüpfen Sie eine Elastic IP-Adresse mit einer Amazon EC2 EC2-Instance.

Das Szenario

Eine Elastic IP-Adresse ist eine statische IP-Adresse, die für dynamisches Cloud Computing konzipiert ist. Ihrem AWS Konto ist eine Elastic IP-Adresse zugeordnet. Es ist eine öffentliche IP-Adresse, die aus dem Internet erreichbar ist. Wenn Ihre Instance keine öffentliche IP-Adresse hat, können Sie eine Elastic IP-Adresse mit der Instance verknüpfen, damit diese mit dem Internet kommunizieren kann.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um mehrere Amazon EC2 EC2-Operationen mit Elastic IP-Adressen durchzuführen. Die Module Node.js verwenden das SDK für JavaScript die Verwaltung von Elastic IP-Adressen mithilfe der folgenden Methoden der Amazon EC2 EC2-Client-Klasse:

- `describeAddresses`
- `allocateAddress`
- `associateAddress`
- `releaseAddress`

Weitere Informationen zu Elastic IP-Adressen in Amazon EC2 finden Sie unter Elastic IP Addresses im Amazon EC2 User Guide oder Elastic IP Addresses im Amazon EC2 User Guide.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

- Erstellen Sie eine Amazon EC2 EC2-Instance. Weitere Informationen zur Erstellung von Amazon EC2 EC2-Instances finden Sie unter Amazon EC2 EC2-Instances im Amazon EC2 EC2-Benutzerhandbuch oder Amazon EC2 EC2-Instances im Amazon EC2 EC2-Benutzerhandbuch.

Beschreiben von Elastic IP-Adressen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_describeaddresses.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das als Parameter übergeben werden soll, das die Adressen filtert, die von denen in Ihrer VPC zurückgegeben werden. Um die Beschreibungen aller Ihrer Elastic IP-Adressen abzurufen, lassen Sie einen Filter von der Parameter-JSON weg. Rufen Sie dann die `describeAddresses` Methode des Amazon EC2-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_describeaddresses.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Zuweisen und Zuordnen einer Elastic IP-Adresse zu einer Amazon EC2 EC2-Instance

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_allocateaddress.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie ein JSON-Objekt für die Parameter, die zum Zuweisen einer Elastic IP-Adresse verwendet werden, mit dem in diesem Fall angegeben wird, dass es sich bei der `Domain` um eine VPC handelt. Rufen Sie die `allocateAddress` Methode des Amazon EC2-Serviceobjekts auf.

Bei erfolgreichem Aufruf verfügt der `data`-Parameter der Callback-Funktion über eine `AllocationId`-Eigenschaft, die die zugewiesene Elastic IP-Adresse identifiziert.

Erstellen Sie ein JSON-Objekt für die Parameter, die verwendet werden, um einer Amazon EC2 EC2-Instance eine Elastic IP-Adresse zuzuordnen, einschließlich der Adresse `AllocationId` von der neu zugewiesenen Adresse und `InstanceId` der der Amazon EC2 EC2-Instance. Rufen Sie dann die `associateAddresses` Methode des Amazon EC2-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_allocateaddress.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Freigeben einer Elastic IP-Adresse

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ec2_releaseaddress.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon EC2 zuzugreifen, erstellen Sie ein `AWS.EC2` Serviceobjekt. Erstellen Sie ein JSON-Objekt für die Parameter, die zum Freigeben einer Elastic IP-Adresse verwendet werden, die in diesem Fall die `AllocationId` für die Elastic IP-Adresse angeben. Durch die Freigabe einer Elastic IP-Adresse wird sie auch von jeder Amazon EC2 EC2-Instance getrennt. Rufen Sie die `releaseAddress` Methode des Amazon EC2-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_releaseaddress.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Beispiele für AWS Elemental MediaConvert

AWS Elemental MediaConvert ist ein dateibasierter Transcodierungsservice für Videos, der über Ausstrahlungsfunktionen verfügt. Sie können damit Inhalte für die Übertragung und für die video-on-demand (VOD-) Übertragung über das Internet erstellen. Weitere Informationen finden Sie im AWS Elemental MediaConvert-Benutzerhandbuch.

Die JavaScript API für MediaConvert wird über die `AWS.MediaConvert` Client-Klasse verfügbar gemacht. Weitere Informationen finden Sie `Class: AWS.MediaConvert` in der API-Referenz.

Themen

- Holen Sie sich Ihren regionsspezifischen Endpunkt für MediaConvert
- Transcodierungsaufträge erstellen und verwalten in MediaConvert
- Verwenden von Jobvorlagen in MediaConvert

Holen Sie sich Ihren regionsspezifischen Endpunkt für MediaConvert

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie Ihren regionsspezifischen Endpunkt von ab. MediaConvert

Das Szenario

In diesem Beispiel verwenden Sie ein Modul Node.js, um Ihren regionsspezifischen Endpunkt aufzurufen MediaConvert und abzurufen. Sie können Ihre Endpunkt-URL vom Standardendpunkt des Dienstes abrufen und benötigen Ihren regionsspezifischen Endpunkt daher noch nicht. Der Code verwendet das SDK JavaScript , um diesen Endpunkt mithilfe dieser Methode der MediaConvert Client-Klasse abzurufen:

- `describeEndpoints`

> **Important**
>
> Der standardmäßige Node.js-HTTP/HTTPS-Agent erstellt eine neue TCP-Verbindung für jede neue Anforderung. Um die Kosten für den Aufbau einer neuen Verbindung zu vermeiden, werden TCP-Verbindungen AWS SDK for JavaScript wiederverwendet. Weitere Informationen finden Sie unter Wiederverwenden von Verbindungen mit Keep-Alive in Node.js.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine IAM-Rolle, die MediaConvert Zugriff auf Ihre Eingabedateien und die Amazon S3 S3-Buckets gewährt, in denen Ihre Ausgabedateien gespeichert sind. Einzelheiten finden Sie unter Einrichten von IAM-Berechtigungen im AWS Elemental MediaConvert Benutzerhandbuch.

Abrufen Ihrer Endpunkt-URL

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_getendpoint.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren.

Erstellen Sie ein Objekt, um die leeren Anforderungsparameter für die `describeEndpoints` Methode der `AWS.MediaConvert` Client-Klasse zu übergeben. Um die `describeEndpoints`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines MediaConvert -Serviceobjekts und übergeben die Parameter. Verarbeiten Sie die Antwort im Promise-Rückruf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node emc_getendpoint.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Transcodierungsaufträge erstellen und verwalten in MediaConvert

Dieses Node.js-Codebeispiel zeigt:

- So geben Sie den regionsspezifischen Endpunkt an, mit dem verwendet werden soll. MediaConvert
- So erstellen Sie Transcodierungsaufträge in. MediaConvert
- So stornieren Sie einen Trancodierungsauftrag.
- So rufen Sie die JSON für einen abgeschlossenen Transcodierungsauftrag ab.
- So rufen Sie ein JSON-Array für bis zu 20 der zuletzt erstellten Aufträge ab.

Das Szenario

In diesem Beispiel verwenden Sie ein Modul Node.js zum Aufrufen, um Transcodierungsaufträge MediaConvert zu erstellen und zu verwalten. Der Code verwendet JavaScript dazu das SDK, indem er die folgenden Methoden der MediaConvert Client-Klasse verwendet:

- `createJob`
- `cancelJob`
- `getJob`
- `listJobs`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen und konfigurieren Sie Amazon S3 S3-Buckets, die Speicherplatz für Auftragseingabedateien und Ausgabedateien bereitstellen. Einzelheiten finden Sie unter Create Storage for Files im AWS Elemental MediaConvertBenutzerhandbuch.
- Laden Sie das Eingabevideo in den Amazon S3 S3-Bucket hoch, den Sie für den Eingabespeicher bereitgestellt haben. Eine Liste der unterstützten Eingabe-Videocodecs und Container finden Sie im Benutzerhandbuch unter Unterstützte Eingabecodecs und Container. AWS Elemental MediaConvert

- Erstellen Sie eine IAM-Rolle, die MediaConvert Zugriff auf Ihre Eingabedateien und die Amazon S3 S3-Buckets gewährt, in denen Ihre Ausgabedateien gespeichert sind. Einzelheiten finden Sie unter Einrichten von IAM-Berechtigungen im AWS Elemental MediaConvert Benutzerhandbuch.

Konfigurieren des SDKs

Konfigurieren Sie das SDK für, JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt. Da für jedes Konto benutzerdefinierte Endpunkte MediaConvert verwendet werden, müssen Sie auch die `AWS.MediaConvert` Client-Klasse so konfigurieren, dass sie Ihren regionsspezifischen Endpunkt verwendet. Setzen Sie zu diesem Zweck den `endpoint`-Parameter auf `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" };
```

Definieren von einfachen Transcodierungsaufträgen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_createjob.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Erstellen Sie die JSON für die Definition der Transcodierungsauftragsparameter.

Diese Parameter sind sehr detailliert. Sie können die AWS Elemental MediaConvertKonsole verwenden, um die JSON-Jobparameter zu generieren, indem Sie Ihre Job-Einstellungen in der Konsole auswählen und dann am Ende des Job-Abschnitts die Option Job-JSON anzeigen auswählen. Dieses Beispiel enthält die JSON für einen einfachen Auftrag.

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

Erstellen eines Transcodierungsauftrags

Nachdem Sie die Auftragsparameter-JSON erstellt haben, rufen Sie die `createJob`-Methode auf, indem Sie ein Promise für den Aufruf eines `AWS.MediaConvert`-Serviceobjekts erstellen, das die Parameter übergibt. Verarbeiten Sie anschließend die response im Promise-Callback. Die ID des erstellten Auftrags wird in den `data` der Antwort zurückgegeben.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node emc_createjob.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Stornieren eines Transcodierungsauftrags

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_canceljob.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Erstellen Sie die JSON, die die ID des zu stornierenden Auftrags enthält. Rufen Sie dann die `cancelJob`-Methode auf, indem Sie ein Promise für den Aufruf eines `AWS.MediaConvert`-Serviceobjekts erstellen, das die Parameter übergibt. Verarbeiten Sie die Antwort im Promise-Rückruf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ec2_canceljob.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Auflisten der letzten Transcodierungsaufträge

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_listjobs.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren.

Erstellen Sie die Parameter-JSON einschließlich der Werte, um festzulegen, ob die Liste in `ASCENDING` oder `DESCENDING` Reihenfolge sortiert wird. Außerdem legen Sie hier den ARN der zu prüfenden Auftragswarteschlange sowie den Status der einzubeziehenden Aufträge fest. Rufen Sie dann die `listJobs`-Methode auf, indem Sie ein Promise für den Aufruf eines `AWS.MediaConvert`-Serviceobjekts erstellen, das die Parameter übergibt. Verarbeiten Sie die Antwort im Promise-Rückruf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node emc_listjobs.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verwenden von Jobvorlagen in MediaConvert

Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie MediaConvert Jobvorlagen.
- So verwenden Sie eine Auftragsvorlage zum Erstellen eines Transcodierungsauftrags.
- So listen Sie alle Ihre Auftragsvorlagen auf.
- So löschen Sie Auftragsvorlagen.

Das Szenario

Die JSON-Datei, die für die Erstellung eines Transcodierungsauftrags erforderlich MediaConvert ist, ist detailliert und enthält eine große Anzahl von Einstellungen. Sie können die Auftragserstellung

erheblich vereinfachen, indem Sie zweifelsfrei funktionierende Einstellungen in einer Auftragsvorlage speichern, die zum Erstellen nachfolgender Aufträge verwendet werden kann. In diesem Beispiel verwenden Sie ein Modul Node.js zum Aufrufen, um Jobvorlagen MediaConvert zu erstellen, zu verwenden und zu verwalten. Der Code verwendet JavaScript dazu das SDK, indem er die folgenden Methoden der MediaConvert Client-Klasse verwendet:

- `createJobTemplate`
- `createJob`
- `deleteJobTemplate`
- `listJobTemplates`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Installieren Sie Node.js. Weitere Informationen finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine IAM-Rolle, die MediaConvert Zugriff auf Ihre Eingabedateien und die Amazon S3 S3-Buckets gewährt, in denen Ihre Ausgabedateien gespeichert sind. Einzelheiten finden Sie unter Einrichten von IAM-Berechtigungen im AWS Elemental MediaConvert Benutzerhandbuch.

Erstellen einer Auftragsvorlage

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_create_jobtemplate.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren.

Geben Sie die Parameter-JSON für die Vorlagenerstellung an. Sie können die meisten JSON-Parameter aus einem vorherigen erfolgreichen Job verwenden, um die `Settings`-Werte in der Vorlage anzugeben. In diesem Beispiel werden die Aufgabeneinstellungen aus Transcodierungsaufträge erstellen und verwalten in MediaConvert verwendet.

Rufen Sie die `createJobTemplate`-Methode auf, indem Sie ein Promise für den Aufruf eines `AWS.MediaConvert`-Serviceobjekts erstellen und die Parameter übergeben. Verarbeiten Sie anschließend die response im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node emc_create_jobtemplate.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Erstellen eines Transcodierungsauftrags aus einer Auftragsvorlage

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_template_createjob.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren.

Erstellen Sie die Auftragserstellungsparameter-JSON, einschließlich dem Namen der zu verwendenden Auftragsvorlage und der zu verwendenden `Settings`, die spezifisch für den Auftrag sind, den Sie erstellen. Rufen Sie dann die `createJobs`-Methode auf, indem Sie ein Promise für den Aufruf eines `AWS.MediaConvert`-Serviceobjekts erstellen, das die Parameter übergibt. Verarbeiten Sie die Antwort im Promise-Rückruf.

```
// 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);
```

```
  }
);
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node emc_template_createjob.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Auflisten Ihrer Jobvorlagen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_listtemplates.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren.

Erstellen Sie ein Objekt, um die Anfrageparameter für die `listTemplates`-Methode der `AWS.MediaConvert`-Client-Klasse zu übergeben. Schließen Sie Werte ein, um zu bestimmen, welche Vorlagen gelistet werden sollen (`NAME`, `CREATION_DATE`, `SYSTEM`), wie viele und deren Sortierreihenfolge. Um die `listTemplates` Methode aufzurufen, erstellen Sie ein Versprechen für den Aufruf eines MediaConvert Serviceobjekts, indem Sie die Parameter übergeben. Verarbeiten Sie anschließend die response im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node emc_listtemplates.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen einer Auftragsvorlage

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_deletetemplate.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren.

Erstellen Sie ein Objekt, um den Namen der Auftragsvorlage, die Sie löschen möchten, als Parameter für die `deleteJobTemplate`-Methode der `AWS.MediaConvert`-Client-Klasse zu übergeben. Um die `deleteJobTemplate` Methode aufzurufen, erstellen Sie ein Versprechen für den Aufruf eines MediaConvert Serviceobjekts, indem Sie die Parameter übergeben. Verarbeiten Sie die Antwort im Promise-Rückruf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node emc_deletetemplate.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Beispiele für Amazon S3 Glacier

Amazon S3 Glacier ist ein sicherer Cloud-Speicherservice für Datenarchivierung und Langzeitsicherung. Der Service ist für Daten optimiert, auf die selten zugegriffen wird und für die eine Abrufdauer von mehreren Stunden geeignet ist.

Die JavaScript API für Amazon S3 Glacier wird über die `AWS.Glacier` Client-Klasse verfügbar gemacht. Weitere Informationen zur Verwendung der S3 Glacier-Clientklasse finden Sie `Class: AWS.Glacier` in der API-Referenz.

Themen

- Einen S3 Glacier Vault erstellen
- Ein Archiv auf S3 Glacier hochladen

- Einen mehrteiligen Upload auf S3 Glacier durchführen

Einen S3 Glacier Vault erstellen

Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie einen Tresor mit der `createVault` Methode des Amazon S3 Glacier-Serviceobjekts.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Den Tresor erstellen

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

```
});
```

Ein Archiv auf S3 Glacier hochladen

Dieses Node.js-Codebeispiel zeigt:

- So laden Sie mithilfe der `uploadArchive` Methode des S3 Glacier-Serviceobjekts ein Archiv auf Amazon S3 Glacier hoch.

Im folgenden Beispiel wird mithilfe der `uploadArchive` Methode des S3 Glacier-Serviceobjekts ein einzelnes `Buffer` Objekt als ganzes Archiv hochgeladen.

In diesem Beispiel wird davon ausgegangen, dass Sie bereits einen Tresor mit dem Namen `YOUR_VAULT_NAME` erstellt haben. Das SDK berechnet automatisch die Struktur-Hash-Prüfsumme für die hochgeladenen Daten. Sie können dies jedoch außer Kraft setzen, indem Sie Ihren eigenen Prüfsummenparameter übergeben:

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Das Archiv hochladen

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

Einen mehrteiligen Upload auf S3 Glacier durchführen

Das folgende Beispiel erstellt mithilfe der `initiateMultipartUpload` Methode des Amazon S3 `Buffer` Glacier-Serviceobjekts einen mehrteiligen Upload aus 1-Megabyte-Blöcken eines Objekts.

In diesem Beispiel wird davon ausgegangen, dass Sie bereits einen Tresor mit dem Namen `YOUR_VAULT_NAME` erstellt haben. Ein vollständiger SHA-256-Struktur-Hash wird manuell mit der `computeChecksums`-Methode berechnet.

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

AWSIAM-Beispiele

AWS Identity and Access Management(IAM) ist ein Webservice, der es Kunden von Amazon Web Services ermöglicht, Benutzer und Benutzerberechtigungen in AWS zu verwalten. Der Service richtet sich an Unternehmen mit mehreren Benutzern oder Systemen in der Cloud, die AWS Produkte verwenden. Mit IAM können Sie Benutzer, Sicherheitsanmeldeinformationen wie Zugriffsschlüssel und Berechtigungen, mit denen gesteuert wird, auf welche AWS Ressourcen Benutzer zugreifen können, zentral verwalten.

Die JavaScript API für IAM wird über die `AWS.IAM` Client-Klasse verfügbar gemacht. Weitere Informationen zur Verwendung der IAM-Clientklasse finden Sie `Class: AWS.IAM`in der API-Referenz.

Themen

- Verwalten von IAM-Benutzern
- Arbeiten mit IAM-Richtlinien
- Verwalten von IAM-Zugriffsschlüsseln
- Arbeiten mit IAM-Serverzertifikaten
- Verwalten von IAM-Konto-Aliassen

Verwalten von IAM-Benutzern

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie eine Liste von IAM-Benutzern ab.
- das Erstellen und Löschen von Benutzern.
- das Aktualisieren eines Benutzernamens.

Das Szenario

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Benutzer in IAM zu erstellen und zu verwalten. Die Module Node.js verwenden das SDK, JavaScript um Benutzer mithilfe der folgenden Methoden der `AWS.IAM` Client-Klasse zu erstellen, zu löschen und zu aktualisieren:

- `createUser`
- `listUsers`
- `updateUser`
- `getUser`
- `deleteUser`

Weitere Informationen zu IAM-Benutzern finden Sie unter IAM-Benutzer im IAM-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Erstellen eines Benutzers

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_createuser.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein Serviceobjekt. `AWS.IAM` Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern, einschließlich des Benutzernamens, den Sie für den neuen Benutzer als Befehlszeilenparameter verwenden möchten.

Rufen Sie die `getUser`-Methode des `AWS.IAM`-Serviceobjekts auf, um festzustellen, ob der Benutzername bereits vorhanden ist. Wenn der Benutzername gegenwärtig nicht vorhanden ist, rufen Sie die `createUser`-Methode auf, um ihn zu erstellen. Falls der Name bereits vorhanden ist, schreiben Sie eine entsprechende Nachricht an die Konsole.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_createuser.js USER_NAME
```

Diesen Beispielcode finden Sie hier unter GitHub.

Auflisten von Benutzern in Ihrem Konto

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_listusers.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Auflisten Ihrer Benutzer enthält. Begrenzen Sie die zurückgegebene Anzahl, indem Sie den `MaxItems`-Parameter auf 10 setzen. Rufen Sie die `listUsers`-Methode des `AWS.IAM`-Serviceobjekts auf. Schreiben Sie den Namen und das Erstellungsdatum des ersten Benutzers an die Konsole.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_listusers.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Aktualisieren eines Benutzernamens

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_updateuser.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern, um Ihre Benutzer aufzulisten. Geben Sie sowohl die aktuellen als auch die neuen Benutzernamen als Befehlszeilenparameter an. Rufen Sie die `updateUser`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um dieses Beispiel auszuführen, geben Sie die folgenden Informationen in die Befehlszeile ein, definieren Sie dabei den aktuellen Benutzerenamen gefolgt vom neuen Benutzernamen.

```
node iam_updateuser.js ORIGINAL_USERNAME NEW_USERNAME
```

Diesen Beispielcode finden Sie hier unter GitHub.

Löschen eines Benutzers

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_deleteuser.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie

ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern, einschließlich des Benutzernamens, den Sie als Befehlszeilenparameter löschen möchten.

Rufen Sie die `getUser`-Methode des `AWS.IAM`-Serviceobjekts auf, um festzustellen, ob der Benutzername bereits vorhanden ist. Falls der Benutzername derzeit nicht vorhanden ist, schreiben Sie eine entsprechende Nachricht an die Konsole. Wenn der Benutzer vorhanden ist, rufen Sie die `deleteUser`-Methode auf, um ihn zu löschen.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_deleteuser.js USER_NAME
```

Diesen Beispielcode finden Sie hier unter GitHub.

Arbeiten mit IAM-Richtlinien

Dieses Node.js-Codebeispiel zeigt:

- So erstellen und löschen Sie IAM-Richtlinien.
- So hängen Sie IAM-Richtlinien an Rollen an und trennen sie von ihnen.

Das Szenario

Sie können einem Benutzer Berechtigungen erteilen, indem Sie eine Richtlinie erstellen, d. h., ein Dokument, in dem die Aktionen aufgeführt sind, die ein Benutzer ausführen kann, sowie die Ressourcen, auf die sich diese Aktionen auswirken können. Alle Aktionen oder Ressourcen, die nicht explizit erlaubt sind, werden standardmäßig verweigert. Richtlinien können erstellt und an Benutzer, Benutzergruppen, von Benutzern übernommene Rollen und Ressourcen angehängt werden.

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Richtlinien in IAM zu verwalten. Die Module Node.js verwenden das SDK JavaScript zum Erstellen und Löschen von Richtlinien sowie zum Anhängen und Trennen von Rollenrichtlinien mithilfe der folgenden Methoden der `AWS.IAM` Client-Klasse:

- `createPolicy`
- `getPolicy`
- `listAttachedRolePolicies`
- `attachRolePolicy`
- `detachRolePolicy`

Weitere Informationen zu IAM-Benutzern finden Sie im IAM-Benutzerhandbuch unter Überblick über die Zugriffsverwaltung: Berechtigungen und Richtlinien.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine IAM-Rolle, der Sie Richtlinien zuordnen können. Weitere Informationen zum Erstellen von Rollen finden Sie unter Creating IAM-Rollen im IAM-Benutzerhandbuch.

Eine IAM-Richtlinie erstellen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_createpolicy.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie zwei JSON-Objekte, eines mit dem Richtliniendokument, das Sie erstellen möchten, und das andere mit den erforderlichen Parametern, die zum Erstellen der Richtlinie erforderlich sind. Hierzu gehören die Richtlinien-JSON und der Name, den Sie der Richtlinie geben möchten. Stellen Sie sicher, dass Sie das JSON-Objekt der Richtlinie in den Parametern in eine Zeichenfolge umwandeln. Rufen Sie die `createPolicy`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_createpolicy.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Eine IAM-Richtlinie erhalten

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_getpolicy.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern, um eine Richtlinie abzurufen. Dabei handelt es sich um den ARN der Richtlinie, die Sie abrufen möchten. Rufen Sie die `getPolicy`-Methode des `AWS.IAM`-Serviceobjekts auf. Schreiben Sie die Richtlinienbeschreibung an die Konsole.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_getpolicy.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Anfügen einer verwalteten Rollenrichtlinie

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_attachrolepolicy.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die Parameter enthält, die zum Abrufen einer Liste der verwalteten IAM-Richtlinien erforderlich sind, die einer Rolle zugeordnet sind. Diese Liste besteht aus dem Namen der Rolle. Geben Sie den Rollennamen als Befehlszeilen-Parameter ein. Rufen Sie die `listAttachedRolePolicies`-Methode des `AWS.IAM`-Objekts auf. Diese gibt ein Array von verwalteten Richtlinien an die Callback-Funktion zurück.

Überprüfen Sie die Array-Mitglieder, um festzustellen, ob die Richtlinie, die Sie der Rolle anfügen möchten, bereits zugeordnet ist. Wenn die Richtlinie nicht verknüpft ist, rufen Sie die `attachRolePolicy`-Methode auf, um die Richtlinie anzufügen.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_attachrolepolicy.js IAM_ROLE_NAME
```

Trennen einer verwalteten Rollenrichtlinie

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_detachrolepolicy.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen

Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die Parameter enthält, die zum Abrufen einer Liste der verwalteten IAM-Richtlinien erforderlich sind, die einer Rolle zugeordnet sind. Diese Liste besteht aus dem Namen der Rolle. Geben Sie den Rollennamen als Befehlszeilen-Parameter ein. Rufen Sie die `listAttachedRolePolicies`-Methode des `AWS.IAM`-Objekts auf. Diese gibt ein Array von verwalteten Richtlinien an die Callback-Funktion zurück.

Überprüfen Sie die Array-Mitglieder, um festzustellen, ob die Richtlinie, die Sie von der Rolle trennen möchten, angefügt ist. Wenn die Richtlinie angefügt ist, rufen Sie die `detachRolePolicy`-Methode auf, um die Richtlinie zu trennen.

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

```
  }
});
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_detachrolepolicy.js IAM_ROLE_NAME
```

Verwalten von IAM-Zugriffsschlüsseln

Dieses Node.js-Codebeispiel zeigt:

- das Verwalten der Zugriffsschlüssel Ihrer Benutzer.

Das Szenario

Benutzer benötigen ihre eigenen Zugriffstasten, um programmgesteuerte Aufrufe AWS vom SDK für tätigen zu können. JavaScript Um diese Anforderung zu erfüllen, können Sie Zugriffsschlüssel (Zugriffsschlüssel-IDs und geheime Zugriffsschlüssel) für IAM-Benutzer erstellen, ändern, anzeigen oder rotieren. Wenn Sie einen Zugriffsschlüssel erstellen, ist der Status standardmäßig `Active`, was bedeutet, dass der Benutzer den Zugriffsschlüssel für API-Aufrufe verwenden kann.

In diesem Beispiel werden eine Reihe von Node.js -Modulen zur Verwaltung von Zugriffsschlüsseln in IAM verwendet. Die Module Node.js verwenden das SDK JavaScript zur Verwaltung von IAM-Zugriffsschlüsseln mithilfe der folgenden Methoden der `AWS.IAM` Client-Klasse:

- createAccessKey
- listAccessKeys
- getAccessKeyLastUsed
- updateAccessKey
- deleteAccessKey

Weitere Informationen zu IAM-Zugriffsschlüsseln finden Sie unter Access Keys im IAM-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Erstellen von Zugriffsschlüsseln für einen Benutzer

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_createaccesskeys.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den Parametern, die zum Erstellen neuer Zugriffsschlüssel erforderlich sind, einschließlich des Namens des IAM-Benutzers. Rufen Sie die `createAccessKey`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Stellen Sie sicher, die zurückgegebenen Daten an eine Textdatei weiterzuleiten, um den geheimen Schlüssel nicht zu verlieren, da dieser nur einmal bereitgestellt werden kann.

```
node iam_createaccesskeys.js > newuserkeys.txt
```

Diesen Beispielcode finden Sie hier unter GitHub.

Auflisten der Zugriffsschlüssel eines Benutzers

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_listaccesskeys.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den Parametern, die zum Abrufen der Zugriffsschlüssel des Benutzers erforderlich sind. Dazu gehören der Name des IAM-Benutzers und optional die maximale Anzahl von Zugriffsschlüsselpaaren, die Sie auflisten möchten. Rufen Sie die `listAccessKeys`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_listaccesskeys.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Abrufen der letzten Verwendung des Zugriffsschlüssels

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_accesskeylastused.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern, um neue Zugriffsschlüssel zu erstellen. Dies ist die Zugriffsschlüssel-ID, für die Sie die Informationen über die letzte Verwendung möchten. Rufen Sie die `getAccessKeyLastUsed`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_accesskeylastused.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Aktualisieren des Zugriffsschlüsselstatus

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_updateaccesskey.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern,

um den Status der Zugriffsschlüssel zu aktualisieren. Dazu gehören die Zugriffsschlüssel-ID und der aktualisierte Status. Der Status kann sowohl `Active` oder `Inactive` sein. Rufen Sie die `updateAccessKey`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_updateaccesskey.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Löschen von Zugriffsschlüsseln

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_deleteaccesskey.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern, um Zugriffsschlüssel zu löschen. Dazu gehören die Zugriffsschlüssel-ID und der Name des Benutzers. Rufen Sie die `deleteAccessKey`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_deleteaccesskey.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Arbeiten mit IAM-Serverzertifikaten

Dieses Node.js-Codebeispiel zeigt:

- die Handhabung grundlegender Aufgaben für das Verwalten von Serverzertifikaten für HTTPS-Verbindungen.

Das Szenario

Um HTTPS-Verbindungen zu Ihrer Website oder Anwendung zu aktivierenAWS, benötigen Sie ein SSL/TLS-Serverzertifikat. Um ein Zertifikat zu verwenden, das Sie von einem externen Anbieter mit

Ihrer Website oder Anwendung bezogen habenAWS, müssen Sie das Zertifikat in IAM hochladen oder in AWS Certificate Manager importieren.

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Serverzertifikate in IAM zu verarbeiten. Die Module Node.js verwenden das SDK für JavaScript die Verwaltung von Serverzertifikaten mithilfe der folgenden Methoden der `AWS.IAM` Client-Klasse:

- listServerCertificates
- getServerCertificate
- updateServerCertificate
- deleteServerCertificate

Weitere Informationen zu Serverzertifikaten finden Sie unter Arbeiten mit Serverzertifikaten im IAM-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Auflisten Ihrer Serverzertifikate

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_listservercerts.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Rufen Sie die `listServerCertificates`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_listservercerts.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Abrufen eines Serverzertifikats

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_getservercert.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern, um ein Zertifikat zu erhalten. Dazu gehört der Name des gewünschten Serverzertifikats. Rufen Sie die `getServerCertificates`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

```
);
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_getservercert.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Aktualisieren eines Serverzertifikats

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_updateservercert.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern zum Aktualisieren eines Zertifikats, bestehend aus dem Namen des vorhandenen Serverzertifikats sowie dem Namen des neuen Zertifikats. Rufen Sie die `updateServerCertificate`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_updateservercert.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Löschen eines Serverzertifikats

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_deleteservercert.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern, um ein Serverzertifikat zu löschen. Dazu gehört der Name des Zertifikats, das Sie löschen möchten. Rufen Sie die `deleteServerCertificates`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_deleteservercert.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Verwalten von IAM-Konto-Aliassen

Dieses Node.js-Codebeispiel zeigt:

- So verwalten Sie Aliase für Ihre AWS Konto-ID.

Das Szenario

Wenn Sie möchten, dass die URL für Ihre Anmeldeseite Ihren Firmennamen oder eine andere benutzerfreundliche Kennung anstelle Ihrer AWS Konto-ID enthält, können Sie einen Alias für Ihre AWS Konto-ID erstellen. Wenn Sie einen AWS-Konto-Alias erstellen, wird der Alias in die URL Ihrer Anmeldeseite integriert.

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um IAM-Kontoaliase zu erstellen und zu verwalten. Die Module Node.js verwenden das SDK JavaScript zur Verwaltung von Aliasen mithilfe der folgenden Methoden der `AWS.IAM` Client-Klasse:

- createAccountAlias
- listAccountAliases
- deleteAccountAlias

Weitere Informationen zu Aliasnamen für IAM-Konten finden Sie unter Ihre AWS Konto-ID und ihr Alias im IAM-Benutzerhandbuch.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Erstellen eines Konto-Alias

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_createaccountalias.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen

Sie ein Serviceobjekt. `AWS.IAM` Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern zum Erstellen eines Konto-Alias, einschließlich dem Alias, den Sie erstellen möchten. Rufen Sie die `createAccountAlias`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_createaccountalias.js ALIAS
```

Diesen Beispielcode finden Sie hier unter GitHub.

Auflisten von Konto-Aliassen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_listaccountaliases.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern zum Auflisten von Konto-Alias, einschließlich der maximalen Anzahl der zurückzugebenden Elemente. Rufen Sie die `listAccountAliases`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_listaccountaliases.js
```

Diesen Beispielcode finden Sie hier unter GitHub.

Löschen eines Konto-Alias

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `iam_deleteaccountalias.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf IAM zuzugreifen, erstellen Sie ein `AWS.IAM` Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern zum Löschen eines Konto-Alias, einschließlich dem Alias, den Sie löschen möchten. Rufen Sie die `deleteAccountAlias`-Methode des `AWS.IAM`-Serviceobjekts auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node iam_deleteaccountalias.js ALIAS
```

Diesen Beispielcode finden Sie hier unter GitHub.

Beispiel für Amazon Kinesis

Amazon Kinesis ist eine Plattform für das Streamen von DatenAWS, die leistungsstarke Dienste zum Laden und Analysieren von Streaming-Daten bietet und Ihnen auch die Möglichkeit bietet, benutzerdefinierte Streaming-Datenanwendungen für spezielle Anforderungen zu erstellen.

Die JavaScript API für Kinesis wird über die `AWS.Kinesis` Client-Klasse verfügbar gemacht. Weitere Informationen zur Verwendung der Kinesis-Clientklasse finden Sie `Class: AWS.Kinesis`in der API-Referenz.

Themen

- Erfassen des Fortschritts beim Scrollen von Webseiten mit Amazon Kinesis

Erfassen des Fortschritts beim Scrollen von Webseiten mit Amazon Kinesis

Dieses Beispiel eines Browser-Skripts zeigt:

- So erfassen Sie den Scroll-Fortschritt auf einer Webseite mit Amazon Kinesis als Beispiel für Nutzungsmetriken von Streaming-Seiten für spätere Analysen.

Das Szenario

In diesem Beispiel simuliert eine einfache HTML-Seite den Inhalt einer Blogseite. Während der Leser den simulierten Blogbeitrag scrollt, verwendet das Browser-Skript das SDK, JavaScript um die Scrollstrecke auf der Seite aufzuzeichnen und diese Daten mithilfe der `putRecords`Methode der Kinesis-Clientklasse an Kinesis zu senden. Die von Amazon Kinesis Data Streams erfassten Streaming-Daten können dann von Amazon EC2 EC2-Instances verarbeitet und in einem von mehreren Datenspeichern gespeichert werden, darunter Amazon DynamoDB und Amazon Redshift.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Erstellen Sie einen Kinesis-Stream. Sie müssen den Ressourcen-ARN des Streams in einem Browser-Skript einschließen. Weitere Informationen zur Erstellung von Amazon Kinesis Data Streams finden Sie unter Managing Kinesis Streams im Amazon Kinesis Data Streams Developer Guide.
- Erstellen Sie einen Amazon Cognito Cognito-Identitätspool mit aktiviertem Zugriff für nicht authentifizierte Identitäten. Sie müssen die Identitäten-Pool-ID im Code einschließen, um Anmeldeinformationen für das Browser-Skript zu erhalten. Weitere Informationen zu Amazon Cognito-Identitätspools finden Sie unter Identitätspools im Amazon Cognito Developer Guide.
- Erstellen Sie eine IAM-Rolle, deren Richtlinie die Erlaubnis erteilt, Daten an einen Kinesis-Stream zu senden. Weitere Informationen zum Erstellen einer IAM-Rolle finden Sie unter Creating a Role to Delegate Permissions to an AWS Service im IAM-Benutzerhandbuch.

Verwenden Sie die folgende Rollenrichtlinie beim Erstellen der IAM-Rolle.

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

```
      },
      {
         "Effect": "Allow",
         "Action": [
            "kinesis:Put*"
         ],
         "Resource": [
            "STREAM_RESOURCE_ARN"
         ]
      }
   ]
}
```

Die Blogseite

Die HTML für die Blogseite besteht hauptsächlich aus einer Reihe von Absätzen in einem `<div>`-Element. Die scrollbare Höhe von `<div>` wird dazu verwendet, um zu berechnen, wie weit ein Leser während des Lesens durch den Inhalt gescrollt hat. Der HTML-Code enthält auch zwei `<script>`-Elemente. Eines dieser Elemente fügt der Seite das SDK für JavaScript hinzu und das andere fügt das Browser-Skript hinzu, das den Scroll-Fortschritt auf der Seite erfasst und an Kinesis meldet.

```
<!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>
```

Konfigurieren des SDKs

Rufen Sie die für die Konfiguration des SDK erforderlichen Anmeldeinformationen ab, indem Sie die `CognitoIdentityCredentials` Methode aufrufen und die Amazon Cognito Cognito-Identitätspool-ID angeben. Erstellen Sie bei Erfolg das Kinesis-Serviceobjekt in der Callback-Funktion.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter Erfassen des Scroll-Verlaufs-Codes der Webseite.)

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

Erstellen von Scroll-Verläufen

Der Scroll-Verlauf wird mithilfe der Eigenschaften `scrollHeight` und `scrollTop` des `<div>`-Elements berechnet, welches den Inhalt des Blogbeitrags enthält. Jeder Scroll-Datensatz wird in einer Event-Listener-Funktion für das `scroll` Ereignis erstellt und dann zu einer Reihe von Datensätzen hinzugefügt, die regelmäßig an Kinesis übermittelt werden.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter Erfassen des Scroll-Verlaufs-Codes der Webseite.)

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

Aufzeichnungen an Kinesis senden

Wenn das Array Datensätze enthält, werden diese ausstehenden Datensätze einmal pro Sekunde an Kinesis gesendet.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter Erfassen des Scroll-Verlaufs-Codes der Webseite.)

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

Erfassen des Scroll-Verlaufs-Codes der Webseite

Hier ist der Browser-Skriptcode für das Beispiel für den Fortschritt des Scrollvorgangs von Kinesis-Webseiten.

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

Amazon S3-Beispiele

Amazon Simple Storage Service (Amazon S3) ist ein Webservice, der hoch skalierbaren Cloud-Speicher bietet. Amazon S3 bietet benutzerfreundlichen Objektspeicher mit einer einfachen Webservice-Schnittstelle zum Speichern und Abrufen beliebiger Datenmengen von überall im Internet.

Die JavaScript API für Amazon S3 wird über die `AWS.S3` Client-Klasse bereitgestellt. Weitere Informationen zur Verwendung der Amazon S3-Clientklasse finden Sie unter `Class: AWS.S3` in der API-Referenz zu .

Themen

Beispiele für Amazon S3-Browser

Die folgenden Themen zeigen zwei Beispiele dafür, wie die im Browser für die Interaktion mit Amazon S3-Buckets verwendet werden AWS SDK for JavaScript kann.

- Das erste Szenario zeigt ein einfaches Szenario, in dem die vorhandenen Fotos in einem Amazon S3-Bucket von jedem (nicht authentifizierten) Benutzer angezeigt werden können.
- Die zweite zeigt ein komplexeres Szenario, in dem Benutzer Operationen an Fotos im Bucket ausführen dürfen, z. B. Hochladen, Löschen usw.

Themen

Anzeigen von Fotos in einem Amazon S3-Bucket aus einem Browser heraus

Dieses Beispiel eines Browser-Skriptcodes veranschaulicht:

- So erstellen Sie ein Fotoalbum in einem Amazon Simple Storage Service (Amazon S3) -Bucket und ermöglichen es nicht authentifizierten Benutzern, die Fotos anzusehen.

Das Szenario

In diesem Beispiel bietet eine einfache HTML-Seite eine Browser-basierte Anwendung zum Anzeigen der Fotos in einem Fotoalbum. Das Fotoalbum befindet sich in einem Amazon S3 S3-Bucket, in den Fotos hochgeladen werden.

Das Browser-Skript verwendet das SDK für JavaScript die Interaktion mit einem Amazon S3 S3-Bucket. Das Skript verwendet die `listObjects`Methode der Amazon S3 S3-Clientklasse, um Ihnen das Anzeigen der Fotoalben zu ermöglichen.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab.

> **Note**
>
> In diesem Beispiel müssen Sie dieselbe AWS Region sowohl für den Amazon S3-Bucket als auch für den Amazon Cognito Cognito-Identitätspool verwenden.

Erstellen des Buckets

Erstellen Sie in der Amazon S3 S3-Konsole einen Amazon S3 S3-Bucket, in dem Sie Alben und Fotos speichern können. Weitere Informationen zur Verwendung der Konsole zum Erstellen eines S3-Buckets finden Sie unter Creating a Bucket im Amazon Simple Storage Service-Benutzerhandbuch.

Wenn Sie den S3-Bucket erstellen, führen Sie unbedingt die folgenden Schritte aus:

- Notieren Sie sich den Bucket-Namen, damit Sie ihn in der nachfolgenden erforderlichen Aufgabe „Konfigurieren von Rollenberechtigungen“ verwenden können.
- Wählen Sie eine AWS Region aus, in der der Bucket erstellt werden soll. Dies muss dieselbe Region sein, die Sie verwenden werden, um einen Amazon Cognito Cognito-Identitätspool in einer nachfolgenden vorausgesetzten Aufgabe, Create an Identity Pool, zu erstellen.
- Konfigurieren Sie Bucket-Berechtigungen, indem Sie den Abschnitt Berechtigungen für den Zugriff auf Websites einrichten im Amazon Simple Storage Service-Benutzerhandbuch befolgen.

Erstellen eines -Identitäten-Pools

Erstellen Sie in der Amazon Cognito Cognito-Konsole einen Amazon Cognito Cognito-Identitätspool, wie im the section called “Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool” Thema Erste Schritte in einem Browserskript beschrieben.

Notieren Sie sich beim Erstellen des Identitätspools den Namen des Identitätspools sowie den Rollennamen für die nicht authentifizierte Identität.

Konfigurieren der Rollenberechtigungen

Um das Ansehen von Alben und Fotos zu ermöglichen, müssen Sie einer IAM-Rolle des Identitätspools, den Sie gerade erstellt haben, Berechtigungen hinzufügen. Beginnen Sie mit der Erstellung einer Richtlinie wie folgt.

1. Öffnen Sie die IAM-Konsole.
2. Wählen Sie im Navigationsbereich linken Policies (Richtlinien) und dann Create Policy (Richtlinie erstellen) aus.
3. Geben Sie auf der Registerkarte JSON die folgende JSON-Definition ein. Ersetzen Sie dabei aber BUCKET_NAME durch den Namen des Buckets.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME"
         ]
      }
   ]
}
```

4. Wählen Sie die Schaltfläche Review policy (Richtlinie überprüfen) aus, benennen Sie die Richtlinie und stellen Sie (falls gewünscht) eine Beschreibung bereit. Wählen Sie anschließend die Schaltfläche Create policy (Richtlinie erstellen) aus.

Notieren Sie sich unbedingt den Namen, damit Sie ihn finden und der IAM-Rolle später zuordnen können.

Nachdem die Richtlinie erstellt wurde, kehren Sie zur IAM-Konsole zurück. Suchen Sie die IAM-Rolle für die nicht authentifizierte Identität, die Amazon Cognito in der vorherigen vorausgesetzten Aufgabe Create an Identity Pool erstellt hat. Fügen Sie dieser Identität mit der soeben erstellten Richtlinie Berechtigungen hinzu.

Obwohl der Workflow für diese Aufgabe generell dem von the section called “Schritt 2: Fügen Sie der erstellten IAM-Rolle eine Richtlinie hinzu” im Thema Erste Schritte in einem Browser Script entspricht, sind dennoch einige Unterschiede zu beachten:

- Verwenden Sie die neue Richtlinie, die Sie gerade erstellt haben, keine Richtlinie für Amazon Polly.
- Öffnen auf der Seite Attach Permissions (Berechtigungen zuweisen) die Liste Filter policies (Richtlinien filtern), um die neue Richtlinie schnell zu finden, und wählen Sie Customer managed (Vom Kunden verwaltet) aus.

Weitere Informationen zum Erstellen einer IAM-Rolle finden Sie unter Creating a Role to Delegate Permissions to an AWS Service im IAM-Benutzerhandbuch.

Konfigurieren von CORS

Bevor das Browserskript auf den Amazon S3 S3-Bucket zugreifen kann, müssen Sie seine CORS-Konfiguration wie folgt einrichten.

> **Important**
>
> In der neuen S3-Konsole muss die CORS-Konfiguration JSON sein.

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>
```

Erstellen von Alben und Hochladen von Fotos

Da die Benutzer in diesem Beispiel nur die Fotos anzeigen können, die bereits sich bereits im Bucket befinden, müssen Sie einige Alben im Bucket erstellen und Fotos zu ihnen hochladen.

> **Note**
>
> Die Dateinamen für dieses Beispiel müssen mit einem einzelnen Unterstrich ("_") beginnen. Dieses Zeichen ist für die spätere Filterung wichtig. Stellen Sie außerdem sicher, dass die Urheberrechte der Eigentümer des Fotos beachtet werden.

1. Öffnen Sie in der Amazon S3 S3-Konsole den Bucket, den Sie zuvor erstellt haben.
2. Wählen Sie auf der Registerkarte Overview (Übersicht) die Schaltfläche Create folder (Ordner erstellen) aus, um Ordner zu erstellen. Nennen Sie die Ordner in diesem Beispiel „album1“, „album2“ und „album3“.
3. Wählen Sie für album1 und dann album2 den Ordner aus laden Sie anschließend die Fotos wie folgt hoch:

a. Wählen Sie die Schaltfläche Upload (Hochladen) aus.

b. Ziehen Sie die zu verwendenden Fotodateien oder wählen Sie sie aus, und klicken Sie auf Next (Weiter).

c. Wählen Sie unter Manage public permissions (Öffentliche Berechtigungen verwalten) die Option Grant public read access to this object(s) (Diesen Objekten öffentlichen Lesezugriff gewähren) aus.

d. Wählen Sie die Schaltfläche Hochladen (in der linken unteren Ecke) aus.

4. Lassen Sie album3 leer.

Definieren der Webseite

Der HTML-Code für das Foto-Anzeigeprogramm besteht aus einem `<div>`-Element, in dem das Browser-Skript die Anzeigeschnittstelle erstellt. Das erste `<script>`-Element fügt das SDK dem Browser-Skript hinzu. Das zweite `<script>` Element fügt die externe JavaScript Datei hinzu, die den Browser-Skriptcode enthält.

In diesem Beispiel hat die Datei den Namen `PhotoViewer.js`. Sie befindet sich im selben Ordner wie die HTML-Datei. Die aktuelle SDK_VERSION_NUMBER finden Sie in der API-Referenz für das SDK im API-Referenzhandbuch. 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>
```

Konfigurieren des SDKs

Sie erhalten die erforderlichen Anmeldeinformationen zum Konfigurieren des SDK, indem Sie die `CognitoIdentityCredentials`-Methode aufrufen. Sie müssen die Amazon Cognito Cognito-Identitätspool-ID angeben. Erstellen Sie als nächstes ein `AWS.S3`-Serviceobjekt.

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

Der Rest des Codes in diesem Beispiel definiert die folgenden Funktionen zum Sammeln und Präsentieren der Informationen über die Alben und Fotos im Bucket.

- `listAlbums`
- `viewAlbum`

Auflisten von Alben im Bucket

Wenn Sie alle vorhandenen Alben im Bucket aufführen möchten, ruft die `listAlbums`-Funktion der Anwendung die `listObjects`-Methode des `AWS.S3`-Service-Objekts auf. Die Funktion verwendet die `CommonPrefixes`-Eigenschaft, damit der Aufruf nur Objekte zurückgibt, die als Alben verwendet werden (d. h. die Ordner).

Der Rest der Funktion nimmt die Liste der Alben aus dem Amazon S3 S3-Bucket und generiert den HTML-Code, der für die Anzeige der Albumliste auf der Webseite benötigt wird.

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

Anzeigen eines Albums

Um den Inhalt eines Albums im Amazon S3 S3-Bucket anzuzeigen, verwendet die `viewAlbum` Funktion der Anwendung einen Albumnamen und erstellt den Amazon S3 S3-Schlüssel für dieses Album. Im Anschluss daran ruft die Funktion die `listObjects`-Methode des `AWS.S3`-Serviceobjekts auf, um eine Liste aller Objekte (Fotos) im Album zu erhalten.

Der Rest der Funktion übernimmt die Liste der Objekte, die sich im Album befinden, und generiert den erforderlichen HTML-Code für die Darstellung der Fotos auf der Webseite.

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

Fotos in einem Amazon S3 S3-Bucket anzeigen: Vollständiger Code

Dieser Abschnitt enthält den vollständigen JavaScript HTML-Code und den Code für das Beispiel, in dem Fotos in einem Amazon S3 S3-Bucket angezeigt werden können. Weitere Details und Voraussetzungen finden Sie im übergeordneten Abschnitt.

Der HTML-Code für das Beispiel:

```
<!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>
```

Diesen Beispielcode finden Sie hier auf GitHub.

Der Browser-Skript-Code für das Beispiel:

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

Diesen Beispielcode finden Sie hier auf GitHub.

Fotos aus einem Browser auf Amazon S3 hochladen

Dieses Beispiel eines Browser-Skriptcodes veranschaulicht:

- So erstellen Sie eine Browseranwendung, mit der Benutzer Fotoalben in einem Amazon S3 S3-Bucket erstellen und Fotos in die Alben hochladen können.

Das Szenario

In diesem Beispiel bietet eine einfache HTML-Seite eine browserbasierte Anwendung zum Erstellen von Fotoalben in einem Amazon S3 S3-Bucket, in den Sie Fotos hochladen können. In der Anwendung können Sie Fotos und Alben löschen, die von Ihnen hinzugefügt wurden.

Das Browser-Skript verwendet das SDK für JavaScript die Interaktion mit einem Amazon S3 S3-Bucket. Verwenden Sie die folgenden Methoden der Amazon S3 S3-Clientklasse, um die Fotoalbum-Anwendung zu aktivieren:

- `listObjects`

- `headObject`
- `putObject`
- `upload`
- `deleteObject`
- `deleteObjects`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Erstellen Sie in der Amazon S3 S3-Konsole einen Amazon S3 S3-Bucket, in dem Sie die Fotos im Album speichern werden. Weitere Informationen zum Erstellen eines Buckets in der Konsole finden Sie unter Erstellen eines Buckets im Amazon Simple Storage Service-Benutzerhandbuch. Stellen Sie sicher, dass Sie über Berechtigungen zum Lesen und Schreiben für Objekte verfügen. Weitere Informationen zum Einrichten von Bucket-Berechtigungen finden Sie unter Berechtigungen für den Zugriff auf Websites festlegen.
- Erstellen Sie in der Amazon Cognito Cognito-Konsole einen Amazon Cognito Cognito-Identitätspool mithilfe von Federated Identities, wobei der Zugriff für nicht authentifizierte Benutzer in derselben Region wie der Amazon S3 S3-Bucket aktiviert ist. Sie müssen die Identitäten-Pool-ID im Code einschließen, um Anmeldeinformationen für das Browser-Skript zu erhalten. Weitere Informationen zu Amazon Cognito Federated Identities finden Sie unter Amazon Cognito Identity Pools (Federated Identites) im Amazon Cognito Developer Guide.
- Suchen Sie in der IAM-Konsole nach der IAM-Rolle, die von Amazon Cognito für nicht authentifizierte Benutzer erstellt wurde. Fügen Sie die folgende Richtlinie hinzu, um Lese- und Schreibberechtigungen für einen Amazon S3 S3-Bucket zu gewähren. Weitere Informationen zum Erstellen einer IAM-Rolle finden Sie unter Creating a Role to Delegate Permissions to an AWS Service im IAM-Benutzerhandbuch.

  Verwenden Sie diese Rollenrichtlinie für die IAM-Rolle, die von Amazon Cognito für nicht authentifizierte Benutzer erstellt wurde.

  > **Warning**
  >
  > Wenn Sie Zugriff für nicht authentifizierte Benutzer aktivieren, gewähren Sie allen Benutzer auf der ganzen Welt Schreibzugriff auf den Bucket und alle Objekte im Bucket. Dieses Sicherheitsniveau dient in diesem Beispiel dazu, den Schwerpunkt auf den primären Zielen

des Beispiels zu belassen. In vielen Realsituationen wird dringend eine höhere Sicherheit, z. B. die Verwendung authentifizierter Benutzern und Objektbesitz, empfohlen.

```
{
   "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/*"
         ]
      }
   ]
}
```

Konfigurieren von CORS

Bevor das Browserskript auf den Amazon S3 S3-Bucket zugreifen kann, müssen Sie zunächst die CORS-Konfiguration wie folgt einrichten.

Important

In der neuen S3-Konsole muss die CORS-Konfiguration JSON sein.

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>
```

Die Webseite

Der HTML-Code für die Anwendung zum Hochladen von Fotos besteht aus einem -Element, in dem das Browser-Skript die Upload-Benutzeroberfläche erstellt. Das erste <script>-Element fügt das SDK dem Browser-Skript hinzu. Das zweite <script>Element fügt die externe JavaScript Datei hinzu, die den Browser-Skriptcode enthält.

```
<!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>
```

Konfigurieren des SDKs

Rufen Sie die für die Konfiguration des SDK erforderlichen Anmeldeinformationen ab, indem Sie die `CognitoIdentityCredentials` Methode aufrufen und die Amazon Cognito Cognito-Identitätspool-ID angeben. Erstellen Sie als nächstes ein `AWS.S3`-Client-Serviceobjekt.

```
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 },
});
```

Fast der gesamte restliche Code in diesem Beispiel ist in einer Reihe von Funktionen organisiert, die Informationen über die Alben im Bucket sammeln und darstellen, in das Album hochgeladene Fotos laden und anzeigen sowie Fotos und Alben löschen. Diese Funktionen sind:

- `listAlbums`
- `createAlbum`
- `viewAlbum`
- `addPhoto`
- `deleteAlbum`
- `deletePhoto`

Auflisten von Alben im Bucket

Die Anwendung erstellt Alben im Amazon S3 S3-Bucket als Objekte, deren Schlüssel mit einem Schrägstrich beginnen, was darauf hinweist, dass das Objekt als Ordner fungiert. Um alle vorhandenen Alben im Bucket aufzulisten, ruft die `listAlbums`-Funktion der Anwendung die `listObjects`-Methode des `AWS.S3`-Serviceobjekts auf und verwendet gleichzeitig `commonPrefix`, damit der Aufruf nur Objekte wiedergibt, die als Alben verwendet werden.

Der Rest der Funktion nimmt die Liste der Alben aus dem Amazon S3 S3-Bucket und generiert den HTML-Code, der für die Anzeige der Albumliste auf der Webseite benötigt wird. Darüber hinaus ermöglicht die Funktion das Löschen und Öffnen einzelner Alben.

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

Erstellen eines Albums im Bucket

Um ein Album im Amazon S3 S3-Bucket zu erstellen, überprüft die `createAlbum` Funktion der Anwendung zunächst den für das neue Album angegebenen Namen, um sicherzustellen, dass er geeignete Zeichen enthält. Die Funktion bildet dann einen Amazon S3 S3-Objektschlüssel und übergibt ihn an die `headObject` Methode des Amazon S3 S3-Serviceobjekts. Diese Methode gibt die Metadaten des angegebenen Schlüssels zurück, d. h., wenn Daten zurückgegeben werden, ist bereits ein Objekt mit diesem Schlüssel vorhanden.

Wenn das Album nicht bereits vorhanden ist, ruft die Funktion die `putObject`-Methode des `AWS.S3`-Serviceobjekts auf, um das Album zu erstellen. Anschließend ruft sie die `viewAlbum`-Funktion ab, um das neue leere Album anzuzeigen.

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

Anzeigen eines Albums

Um den Inhalt eines Albums im Amazon S3 S3-Bucket anzuzeigen, verwendet die `viewAlbum` Funktion der Anwendung einen Albumnamen und erstellt den Amazon S3 S3-Schlüssel für dieses Album. Im Anschluss daran ruft die Funktion die `listObjects`-Methode des `AWS.S3`-Serviceobjekts ab, um eine Liste aller Objekte (Fotos) im Album zu erhalten.

Der Rest der Funktion übernimmt die Liste der Objekte (Fotos) aus dem Album und generiert den erforderlichen HTML-Code, um die Fotos auf der Webseite anzuzeigen. Außerdem ermöglicht die Funktion das Löschen einzelner Fotos sowie die Navigation zurück zur Albumliste.

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

Hinzufügen von Fotos zu einem Album

Um ein Foto in ein Album im Amazon S3 S3-Bucket hochzuladen, verwendet die `addPhoto` Funktion der Anwendung ein Dateiauswahlelement auf der Webseite, um die hochzuladende Datei zu identifizieren. Sie erstellt dann einen Schlüssel zum Hochladen des Fotos aus dem aktuellen Albumnamen und dem Dateinamen.

Die Funktion ruft die `upload` Methode des Amazon S3-Serviceobjekts auf, um das Foto hochzuladen. Nach dem Hochladen des Fotos, zeigt die Funktion das Album erneut an, damit das hochgeladene Foto erscheint.

```
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);
    }
  );
```

```
}
```

Löschen eines Fotos

Um ein Foto aus einem Album im Amazon S3 S3-Bucket zu löschen, ruft die `deletePhoto` Funktion der Anwendung die `deleteObject` Methode des Amazon S3-Serviceobjekts auf. Dies löscht das Foto, das anhand des `photoKey`-Werts an die Funktion übergeben wurde.

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

Löschen eines Albums

Um ein Album im Amazon S3 S3-Bucket zu löschen, ruft die `deleteAlbum` Funktion der Anwendung die `deleteObjects` Methode des Amazon S3-Serviceobjekts auf.

```
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();
      }
```

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

Fotos auf Amazon S3 hochladen: Vollständiger Code

Dieser Abschnitt enthält den vollständigen JavaScript HTML-Code und den Code für das Beispiel, in dem Fotos in ein Amazon S3 S3-Fotoalbum hochgeladen werden. Weitere Details und Voraussetzungen finden Sie im übergeordneten Abschnitt.

Der HTML-Code für das Beispiel:

```
<!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>
```

Diesen Beispielcode finden Sie hier auf GitHub.

Der Browser-Skript-Code für das Beispiel:

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

Diesen Beispielcode finden Sie hier auf GitHub.

Beispiele für Amazon S3 Node.js

Die folgenden Themen zeigen Beispiele dafür, wie die für die Interaktion mit Amazon S3-Buckets mithilfe von Node.js verwendet werden AWS SDK for JavaScript kann.

Themen

- Erstellen und Verwenden eines Amazon S3-Buckets
- Konfigurieren von Amazon S3-Buckets
- Verwalten von Zugriffsberechtigungen für Amazon S3-Buckets
- Arbeiten mit Amazon S3-Bucket-Richtlinien
- Verwenden eines Amazon S3-Buckets als statischen Web-Host

Erstellen und Verwenden eines Amazon S3-Buckets

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie eine Liste von Amazon S3-Buckets in Ihrem Konto ab und zeigen sie an.
- So erstellen Sie einen Amazon S3-Bucket:
- das Hochladen eines Objekts in einen bestimmten Bucket.

Das Szenario

In diesem Beispiel wird eine Reihe von Node.js-Modulen verwendet, um eine Liste der vorhandenen Amazon S3-Buckets abzurufen, einen Bucket zu erstellen und eine Datei in einen angegebenen Bucket hochzuladen. Diese Node.js-Module verwenden das -SDK für JavaScript , um Informationen aus einem Amazon S3-Bucket abzurufen und Dateien mit diesen Methoden der Amazon S3-Clientklasse in einen Amazon-S3-Bucket hochzuladen:

- `listBuckets`
- `createBucket`
- `listObjects`
- `upload`
- `deleteBucket`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Konfigurieren des SDKs

Konfigurieren Sie das SDK für , JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt.

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

Anzeigen einer Liste von Amazon S3-Buckets

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_listbuckets.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren. Um auf Amazon Simple Storage Service zuzugreifen, erstellen Sie ein -`AWS.S3`Serviceobjekt. Rufen Sie die -`listBuckets`Methode des Amazon S3-Serviceobjekts auf, um eine Liste Ihrer Buckets abzurufen. Der `data`-Parameter der Callback-Funktion verfügt über eine `Buckets`-Eigenschaft, die ein Array von Zuordnungen zur Darstellung der Buckets enthält. Zeigen Sie die Bucket-Liste an, indem Sie sie in der Konsole protokollieren.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_listbuckets.js
```

Dieser Beispielcode finden Sie hier unter GitHub.

Erstellen eines Amazon-S3-Buckets

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_createbucket.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren. Erstellen Sie ein `AWS.S3`-Serviceobjekt. Das Modul ruft ein einzelnes Befehlszeilen-Argument ab, um einen Namen für den neuen Bucket anzugeben.

Fügen Sie eine Variable hinzu, die die Parameter enthält, die zum Aufrufen der `createBucket` Methode des Amazon S3-Serviceobjekts verwendet werden, einschließlich des Namens für den neu erstellten Bucket. Die Rückruffunktion protokolliert den Speicherort des neuen Buckets in der Konsole, nachdem Amazon S3 ihn erfolgreich erstellt hat.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_createbucket.js BUCKET_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Hochladen einer Datei in einen Amazon S3-Bucket

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_upload.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren. Erstellen Sie ein `AWS.S3`-Serviceobjekt. Das Modul umfasst zwei Befehlszeilenargumente, das erste gibt den Ziel-Bucket und das zweite die Datei an, die hochgeladen werden soll.

Erstellen Sie eine Variable mit den Parametern, die zum Aufrufen der `upload` Methode des Amazon S3-Serviceobjekts erforderlich sind. Geben Sie den Namen des Ziel-Buckets im `Bucket`-Parameter an. Der `Key`-Parameter wird auf den Namen der ausgewählten Datei festgelegt. Diesen erhalten Sie mithilfe des Node.js `path`-Moduls. Der `Body`-Parameter wird auf den Inhalt der ausgewählten Datei festgelegt. Diesen erhalten Sie mithilfe von `createReadStream` aus dem Node.js `fs`-Modul.

```
// 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);
```

```
  }
});
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_upload.js BUCKET_NAME FILE_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Auflisten von Objekten in einem Amazon S3-Bucket

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_listobjects.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren. Erstellen Sie ein `AWS.S3`-Serviceobjekt.

Fügen Sie eine Variable hinzu, die die Parameter enthält, die zum Aufrufen der `listObjects` Methode des Amazon S3-Serviceobjekts verwendet werden, einschließlich des Namens des zu lesenden Buckets. Die Callback-Funktion protokolliert eine Liste mit Objekten (Dateien) oder eine Fehlermeldung.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_listobjects.js
```

Dieser Beispielcode finden Sie hier unter GitHub.

Löschen eines Amazon S3-Buckets

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_deletebucket.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren. Erstellen Sie ein `AWS.S3`-Serviceobjekt.

Fügen Sie eine Variable hinzu, die die Parameter enthält, die zum Aufrufen der `createBucket` Methode des Amazon S3-Serviceobjekts verwendet werden, einschließlich des Namens des zu löschenden Buckets. Der Bucket muss leer sein, damit er gelöscht werden kann. Die Callback-Funktion protokolliert eine Erfolgs- oder Fehlermeldung.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_deletebucket.js
```

Dieser Beispielcode finden Sie hier unter GitHub.

Konfigurieren von Amazon S3-Buckets

Dieses Node.js-Codebeispiel zeigt:

- das Konfigurieren der Berechtigungen für das Cross-Origin Resource Sharing (CORS) für einen Bucket.

Das Szenario

In diesem Beispiel werden mehrere Node.js-Module verwendet, um Ihre Amazon S3-Buckets aufzulisten und die CORS- und Bucket-Protokollierung zu konfigurieren. Die Node.js-Module verwenden das SDK für JavaScript , um einen ausgewählten Amazon S3-Bucket mit diesen Methoden der Amazon S3-Clientklasse zu konfigurieren:

- `getBucketCors`
- `putBucketCors`

Weitere Informationen zur Verwendung der CORS-Konfiguration mit einem Amazon S3-Bucket finden Sie unter Cross-Origin Resource Sharing (CORS) im Benutzerhandbuch für Amazon Simple Storage Service.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Konfigurieren des SDKs

Konfigurieren Sie das SDK für , JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt.

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

Abrufen der CORS-Konfiguration für einen Bucket

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_getcors.js`. Das Modul ruft ein einzelnes Befehlszeilenargument ab, um den Bucket anzugeben, dessen CORS-Konfiguration Sie möchten. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren. Erstellen Sie ein `AWS.S3`-Serviceobjekt.

Der einzige Parameter, den Sie beim Abrufen der `getBucketCors`-Methode weiterleiten müssen, ist der Name des ausgewählten Buckets. Wenn der Bucket derzeit über eine CORS-Konfiguration verfügt, wird diese Konfiguration von Amazon S3 als `CORSRules` Eigenschaft des `data` Parameters zurückgegeben, der an die Callback-Funktion übergeben wird.

Wenn der ausgewählte Bucket über keine CORS-Konfiguration verfügt, werden diese Informationen im `error`-Parameter an die Callback-Funktion zurückgegeben.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_getcors.js BUCKET_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Einrichten der CORS-Konfiguration für einen Bucket

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_setcors.js`. Das Modul übernimmt mehrere Befehlszeilenargumente. Das erste gibt den Bucket an, dessen CORS-Konfiguration Sie festlegen möchten. Zusätzliche Argumente listen die HTTP-Methoden auf (POST, GET, PUT, POST, PATCH, DELETE), die Sie für den Bucket zulassen möchten. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein `AWS.S3`-Serviceobjekt. Erstellen Sie als Nächstes ein JSON-Objekt, um die Werte für die CORS-Konfiguration zu umfassen, wie es für die `putBucketCors`-Methode des `AWS.S3`-Serviceobjekts erforderlich ist. Geben Sie `"Authorization"` für den `AllowedHeaders`-Wert und `"*"` für den `AllowedOrigins`-Wert an. Geben Sie den Wert für `AllowedMethods` anfänglich als leeres Array an.

Geben Sie die zulässigen Methoden für das Node.js-Modul als Befehlszeilen-Parameter an, wobei Sie die jeweiligen Methoden hinzufügen, die einem der Parameter entsprechen. Fügen Sie die resultierende CORS-Konfiguration dem Konfigurationsarray hinzu, das im `CORSRules`-Parameter enthalten ist. Legen Sie den Bucket fest, den Sie im `Bucket`-Parameter für CORS konfigurieren möchten.

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

Um das Beispiel auszuführen, müssen Sie den folgenden Befehl, einschließlich einer oder mehrerer HTTP-Methoden, in der Befehlszeile eingeben, wie nachfolgend dargestellt.

```
node s3_setcors.js BUCKET_NAME get put
```

Dieser Beispielcode finden Sie hier unter GitHub.

Verwalten von Zugriffsberechtigungen für Amazon S3-Buckets

Dieses Node.js-Codebeispiel zeigt:

- So können Sie die Zugriffskontrollliste für einen Amazon S3-Bucket abrufen oder festlegen

Das Szenario

In diesem Beispiel wird ein Node.js-Modul dazu verwendet, die Bucket-ACL (Access Control List, Zugriffskontrollliste) für einen ausgewählten Bucket anzuzeigen. Darüber hinaus wird beschrieben, wie Änderungen an der ACL für einen ausgewählten Bucket übernommen werden. Das Modul Node.js verwendet das SDK für JavaScript , um Amazon S3-Bucket-Zugriffsberechtigungen mit diesen Methoden der Amazon S3-Clientklasse zu verwalten:

- `getBucketAcl`
- `putBucketAcl`

Weitere Informationen zu Zugriffskontrolllisten für Amazon S3-Buckets finden Sie unter Verwalten des Zugriffs mit ACLs im Benutzerhandbuch für Amazon Simple Storage Service.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Konfigurieren des SDKs

Konfigurieren Sie das SDK für , JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt.

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

Abrufen der aktuellen Zugriffskontrollliste der Buckets

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_getbucketacl.js`. Das Modul ruft ein einzelnes Befehlszeilenargument ab, um den Bucket festzulegen, dessen ACL-Konfiguration Sie möchten. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren.

Erstellen Sie ein `AWS.S3`-Serviceobjekt. Der einzige Parameter, den Sie beim Abrufen der `getBucketAcl`-Methode weiterleiten müssen, ist der Name des ausgewählten Buckets. Die aktuelle Konfiguration der Zugriffskontrollliste wird von Amazon S3 in dem `data` Parameter zurückgegeben, der an die Rückruffunktion übergeben wird.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_getbucketacl.js BUCKET_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Arbeiten mit Amazon S3-Bucket-Richtlinien

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie die Bucket-Richtlinie eines Amazon S3-Buckets ab.
- So fügen Sie die Bucket-Richtlinie eines Amazon S3-Buckets hinzu oder aktualisieren sie.
- So löschen Sie die Bucket-Richtlinie eines Amazon S3-Buckets.

Das Szenario

In diesem Beispiel wird eine Reihe von Node.js-Modulen verwendet, um eine Bucket-Richtlinie für einen Amazon S3-Bucket abzurufen, festzulegen oder zu löschen. Die Node.js-Module verwenden das SDK für JavaScript , um die Richtlinie für einen ausgewählten Amazon S3-Bucket mit den folgenden Methoden der Amazon S3-Clientklasse zu konfigurieren:

- `getBucketPolicy`
- `putBucketPolicy`
- `deleteBucketPolicy`

Weitere Informationen zu Bucket-Richtlinien für Amazon S3-Buckets finden Sie unter Verwenden von Bucket-Richtlinien und Benutzerrichtlinien im Benutzerhandbuch für Amazon Simple Storage Service.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Konfigurieren des SDKs

Konfigurieren Sie das SDK für , JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt.

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

Abrufen der aktuellen Bucket-Richtlinie

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_getbucketpolicy.js`. Das Modul ruft ein einzelnes Befehlszeilenargument ab, um den Bucket anzugeben, dessen Richtlinie Sie möchten. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren.

Erstellen Sie ein `AWS.S3`-Serviceobjekt. Der einzige Parameter, den Sie beim Abrufen der `getBucketPolicy`-Methode weiterleiten müssen, ist der Name des ausgewählten Buckets. Wenn der Bucket derzeit über eine Richtlinie verfügt, wird diese Richtlinie von Amazon S3 in dem `data` Parameter zurückgegeben, der an die Rückruffunktion übergeben wird.

Wenn der ausgewählte Bucket über keine Richtlinie verfügt, werden diese Informationen im `error`-Parameter an die Callback-Funktion zurückgegeben.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_getbucketpolicy.js BUCKET_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Einrichten einer einfachen Bucket-Richtlinie

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_setbucketpolicy.js`. Das Modul ruft ein einzelnes Befehlszeilenargument ab, um den Bucket anzugeben, dessen Richtlinie Sie anwenden möchten. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein `AWS.S3`-Serviceobjekt. Bucket-Richtlinien sind in JSON-angegeben. Erstellen Sie zunächst ein JSON-Objekt, das alle Werte enthält, die in der Richtlinie festgelegt werden sollen. Eine Ausnahme ist der `Resource`-Wert, über den der Bucket identifiziert wird.

Formatieren Sie die von der Richtlinie benötigte `Resource`-Zeichenfolge und binden Sie den Namen des ausgewählten Buckets darin ein. Fügen Sie diese Zeichenfolge in das JSON-Objekt ein. Bereiten Sie die Parameter für die `putBucketPolicy`-Methode vor, einschließlich dem Namen des Buckets und der in eine Zeichenfolge konvertierten JSON-Richtlinie.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_setbucketpolicy.js BUCKET_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Löschen einer Bucket-Richtlinie

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_deletebucketpolicy.js`. Das Modul ruft ein einzelnes Befehlszeilenargument ab, um den Bucket anzugeben, dessen Richtlinie Sie löschen möchten. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein `AWS.S3`-Serviceobjekt. Der einzige Parameter, den Sie beim Abrufen der `deleteBucketPolicy`-Methode weiterleiten müssen, ist der Name des ausgewählten Buckets.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_deletebucketpolicy.js BUCKET_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Verwenden eines Amazon S3-Buckets als statischen Web-Host

Dieses Node.js-Codebeispiel zeigt:

- So richten Sie einen Amazon S3-Bucket als statischen Webhost ein.

Das Szenario

In diesem Beispiel werden mehrere Node.js-Module verwendet, um einen Ihrer Buckets als statischen Web-Host zu konfigurieren. Die Node.js-Module verwenden das SDK für JavaScript , um einen ausgewählten Amazon S3-Bucket mit diesen Methoden der Amazon S3-Clientklasse zu konfigurieren:

- `getBucketWebsite`
- `putBucketWebsite`
- `deleteBucketWebsite`

Weitere Informationen zur Verwendung eines Amazon S3-Buckets als statischen Webhost finden Sie unter Hosten einer statischen Website auf Amazon S3 im Benutzerhandbuch für Amazon Simple Storage Service.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Konfigurieren des SDKs

Konfigurieren Sie das SDK für , JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
```

```
AWS.config.update({region: 'us-west-2'});
```

Abrufen der aktuellen Konfiguration einer Bucket-Website

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_getbucketwebsite.js`. Das Modul ruft ein einzelnes Befehlszeilenargument ab, um den Bucket anzugeben, dessen Website-Konfiguration Sie möchten. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein `AWS.S3`-Serviceobjekt. Erstellen Sie eine Funktion, die die aktuelle Website-Konfiguration für den Bucket abruft, der in der Bucket-Liste ausgewählt wurde. Der einzige Parameter, den Sie beim Abrufen der `getBucketWebsite`-Methode weiterleiten müssen, ist der Name des ausgewählten Buckets. Wenn der Bucket derzeit über eine Website-Konfiguration verfügt, wird diese Konfiguration von Amazon S3 in dem `data` Parameter zurückgegeben, der an die Rückruffunktion übergeben wird.

Wenn der ausgewählte Bucket über keine Website-Konfiguration verfügt, werden diese Informationen im `err`-Parameter an die Callback-Funktion zurückgegeben.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_getbucketwebsite.js BUCKET_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Einrichten einer Website-Konfiguration eines Buckets

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_setbucketwebsite.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren. Erstellen Sie ein `AWS.S3`-Serviceobjekt.

Erstellen Sie eine Funktion, die eine Bucket-Website-Konfiguration anwendet. Die Konfiguration ermöglicht es dem ausgewählten Bucket als statischer Web-Host zu fungieren. Website-Konfigurationen werden in JSON angegeben. Erstellen Sie zunächst ein JSON-Objekt mit allen Werten, die in der Website-Konfiguration festgelegt werden sollen, mit Ausnahme des `Key`-Werts, der das Fehlerdokument und den `Suffix`-Wert für das Indexdokument identifiziert.

Fügen Sie die Werte der Texteingabe-Elemente in das JSON-Objekt ein. Bereiten Sie die Parameter für die `putBucketWebsite`-Methode vor, einschließlich dem Namen des Buckets und der JSON-Website-Konfiguration.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_setbucketwebsite.js BUCKET_NAME INDEX_PAGE ERROR_PAGE
```

Dieser Beispielcode finden Sie hier unter GitHub.

Löschen der Website-Konfiguration eines Buckets

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `s3_deletebucketwebsite.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor dargestellt konfigurieren. Erstellen Sie ein `AWS.S3`-Serviceobjekt.

Erstellen Sie eine Funktion, die die Website-Konfiguration des ausgewählten Buckets löscht. Der einzige Parameter, den Sie beim Abrufen der `deleteBucketWebsite`-Methode weiterleiten müssen, ist der Name des ausgewählten Buckets.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node s3_deletebucketwebsite.js BUCKET_NAME
```

Dieser Beispielcode finden Sie hier unter GitHub.

Beispiele für Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) ist ein cloudbasierter E-Mail-Versandservice, der digitale Vermarkter und Anwendungsentwickler beim Versenden von Marketing-, Benachrichtigungs- und Transaktions-E-Mails unterstützt. Dabei handelt es sich um einen zuverlässigen, kosteneffektiven Dienst für Unternehmen jeder Größe, die den Kontakt zu ihren Kunden mithilfe von E-Mail aufrechterhalten

Die JavaScript API für Amazon SES wird über die `AWS.SES` Client-Klasse verfügbar gemacht. Weitere Informationen zur Verwendung der Amazon SES SES-Client-Klasse finden Sie `Class: AWS.SES`in der API-Referenz.

Themen

- Verwaltung von Amazon SES SES-Identitäten
- Arbeiten mit E-Mail-Vorlagen in Amazon SES
- Senden von E-Mails mit Amazon SES
- Verwenden von IP-Adressfiltern für den E-Mail-Empfang in Amazon SES
- Verwenden von Empfangsregeln in Amazon SES

Verwaltung von Amazon SES SES-Identitäten

Dieses Node.js-Codebeispiel zeigt:

- So verifizieren Sie E-Mail-Adressen und Domains, die mit Amazon SES verwendet werden.
- So weisen Sie Ihren Amazon SES SES-Identitäten eine IAM-Richtlinie zu.
- So listen Sie alle Amazon SES SES-Identitäten für Ihr AWS Konto auf.
- So löschen Sie Identitäten, die mit Amazon SES verwendet werden.

Eine Amazon SES-Identität ist eine E-Mail-Adresse oder Domain, die Amazon SES zum Senden von E-Mails verwendet. Amazon SES verlangt von Ihnen, Ihre E-Mail-Identitäten zu verifizieren, um zu bestätigen, dass sie Ihnen gehören, und zu verhindern, dass andere sie verwenden.

Einzelheiten zur Verifizierung von E-Mail-Adressen und Domains in Amazon SES finden Sie unter Verifying Email Addresses and Domains in Amazon SES im Amazon Simple Email Service Developer Guide. Informationen zur Sendeautorisierung in Amazon SES finden Sie unter Übersicht über Amazon SES Sending Authorization.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Amazon SES SES-Identitäten zu überprüfen und zu verwalten. Die Module Node.js verwenden das SDK JavaScript zur Überprüfung von E-Mail-Adressen und Domains und verwenden dabei die folgenden Methoden der `AWS.SES` Client-Klasse:

- `listIdentities`
- `deleteIdentity`
- `verifyEmailIdentity`
- `verifyDomainIdentity`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer JSON-Datei mit den Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Konfigurieren des SDKs

Konfigurieren Sie das SDK für, JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');

// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Auflisten Ihrer Identitäten

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Adressen und Domains aufzulisten, die mit Amazon SES verwendet werden sollen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_listidentities.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um den `IdentityType` und andere Parameter für die `listIdentities`-Methode der `AWS.SES`-Client-Klasse zu übergeben. Um die `listIdentities` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SES-Serviceobjekts, indem Sie das Parameterobjekt übergeben.

Verarbeiten Sie anschließend die `response` im Promise-Callback. Die vom Promise-Callback zurückgegebenen `data` enthalten ein Array von Domänen-Identitäten, die vom `IdentityType`-Parameter festgelegt werden.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ses_listidentities.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verifizieren der Identität einer E-Mail-Adresse

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Absender für die Verwendung mit Amazon SES zu verifizieren. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_verifyemailidentity.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Um auf Amazon SES zuzugreifen, erstellen Sie ein `AWS.SES` Serviceobjekt.

Erstellen Sie ein Objekt mit dem sie den `EmailAddress`-Parameter an die `verifyEmailIdentity`-Methode der `AWS.SES`-Client-Klasse übergeben. Um die `verifyEmailIdentity`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Die Domain wird zu Amazon SES hinzugefügt, um verifiziert zu werden.

```
node ses_verifyemailidentity.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verifizieren einer Domänen-Identität

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Domänen für die Verwendung mit Amazon SES zu verifizieren. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_verifydomainidentity.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt mit dem sie den `Domain`-Parameter an die `verifyDomainIdentity`-Methode der `AWS.SES`-Client-Klasse übergeben. Um die `verifyDomainIdentity` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SES-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Die Domain wird zu Amazon SES hinzugefügt, um verifiziert zu werden.

```
node ses_verifydomainidentity.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen von Identitäten

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Adressen oder Domains zu löschen, die mit Amazon SES verwendet werden. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_deleteidentity.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt mit dem sie den `Identity`-Parameter an die `deleteIdentity`-Methode der `AWS.SES`-Client-Klasse übergeben. Um die `deleteIdentity` Methode aufzurufen, erstellen Sie ein Objekt `request` zum Aufrufen eines Amazon SES SES-Serviceobjekts und übergeben dabei die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node ses_deleteidentity.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Arbeiten mit E-Mail-Vorlagen in Amazon SES

Dieses Node.js-Codebeispiel zeigt:

- das Abrufen einer Liste mit all Ihren E-Mail-Vorlagen.
- das Abrufen und Aktualisieren von E-Mail-Vorlagen.
- das Erstellen und Löschen von E-Mail-Vorlagen.

Mit Amazon SES können Sie personalisierte E-Mail-Nachrichten mithilfe von E-Mail-Vorlagen versenden. Einzelheiten zum Erstellen und Verwenden von E-Mail-Vorlagen in Amazon Simple Email Service finden Sie unter Senden personalisierter E-Mails mithilfe der Amazon SES SES-API im Amazon Simple Email Service Developer Guide.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js-Module, um mit E-Mail-Vorlagen zu arbeiten. Die Module Node.js verwenden das SDK JavaScript , um E-Mail-Vorlagen mit den folgenden Methoden der `AWS.SES` Client-Klasse zu erstellen und zu verwenden:

- `listTemplates`
- `createTemplate`

- `getTemplate`
- `deleteTemplate`
- `updateTemplate`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Erstellen einer Anmeldeinformationsdatei finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Auflisten Ihrer E-Mail-Vorlagen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_listtemplates.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, mit dem Sie die Parameter für die `listTemplates`-Methode der `AWS.SES`-Client-Klasse übergeben können. Um die `listTemplates`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES gibt die Liste der Vorlagen zurück.

```
node ses_listtemplates.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Abrufen einer E-Mail-Vorlage

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES abzurufen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_gettemplate.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt mit dem sie den `TemplateName`-Parameter an die `getTemplate`-Methode der `AWS.SES`-Client-Klasse übergeben. Um die `getTemplate`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_gettemplate.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Erstellen einer E-Mail-Vorlage

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_createtemplate.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um die Parameter für die `createTemplate`-Methode der `AWS.SES`-Client-Klasse zu übergeben, einschließlich `TemplateName`, `HtmlPart`, `SubjectPart` und `TextPart`. Um die `createTemplate`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Die Vorlage wird zu Amazon SES hinzugefügt.

```
node ses_createtemplate.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Aktualisieren einer E-Mail-Vorlage

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_updatetemplate.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um die `Template`-Parameterwerte, die Sie in der Vorlage aktualisieren möchten, mit dem erforderlichen `TemplateName`-Parameter an die `updateTemplate`-Methode der `AWS.SES`-Client-Klasse zu übergeben. Um die `updateTemplate`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_updatetemplate.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen einer E-Mail-Vorlage

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_deletetemplate.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um den erforderlichen `TemplateName`-Parameter an die `deleteTemplate`-Methode der `AWS.SES`-Client-Klasse zu übergeben. Um die `deleteTemplate`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_deletetemplate.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Senden von E-Mails mit Amazon SES

Dieses Node.js-Codebeispiel zeigt:

- das Senden einer Text- oder HTML-E-Mail.
- das Senden von E-Mails, die auf einer E-Mail-Vorlage basieren.
- das Senden von Massen-E-Mails, die auf einer E-Mail-Vorlage basieren.

Die Amazon SES SES-API bietet Ihnen zwei verschiedene Möglichkeiten, eine E-Mail zu senden, je nachdem, wie viel Kontrolle Sie über die Zusammensetzung der E-Mail-Nachricht haben möchten: formatiert und roh. Einzelheiten finden Sie unter Senden formatierter E-Mails mit der Amazon SES SES-API und Senden von Roh-E-Mails mit der Amazon SES SES-API.

Das Szenario

In diesem Beispiel verwenden Sie mehrere Node.js-Module, um E-Mails auf verschiedene Weisen zu senden. Die Module Node.js verwenden das SDK JavaScript , um E-Mail-Vorlagen mit den folgenden Methoden der `AWS.SES` Client-Klasse zu erstellen und zu verwenden:

- `sendEmail`
- `sendTemplatedEmail`
- `sendBulkTemplatedEmail`

Erforderliche Aufgaben

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer JSON-Datei mit den Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Voraussetzungen zum Senden von E-Mail-Nachrichten

Amazon SES verfasst eine E-Mail-Nachricht und stellt sie sofort in die Warteschlange für den Versand. Damit Sie mithilfe der `SES.sendEmail`-Methode E-Mails senden können, muss Ihre Nachricht die folgenden Anforderungen erfüllen:

- Sie müssen die Nachricht von einer verifizierten E-Mail-Adresse oder Domäne senden. Wenn Sie versuchen, E-Mails über eine nicht verifizierte Adresse oder Domäne zu senden, führt die Operation zu einem `"Email address not verified"`-Fehler.
- Wenn sich Ihr Konto noch in der Amazon SES Sandbox befindet, können Sie die E-Mails nur an verifizierte Adressen oder Domänen oder E-Mail-Adressen des Amazon SES-Postfachsimulators senden. Weitere Informationen finden Sie unter Verifizieren von E-Mail-Adressen und Domains im Amazon Simple Email Service Developer Guide.
- Die Gesamtgröße der Nachricht, einschließlich Anlagen, muss kleiner als 10 MB sein.
- Die Nachricht muss mindestens eine E-Mail-Adresse für einen Empfänger enthalten. Bei der Empfänger-Adresse kann es sich um eine Empfängeradresse, eine CC: Adresse oder BCC: Adresse handeln. Wenn eine Empfänger-E-Mail-Adresse ungültig ist (d. h., wenn sie nicht das Format `UserName@[SubDomain.]Domain.TopLevelDomain` aufweist), wird die gesamte Nachricht abgelehnt, selbst wenn die Nachricht Empfänger umfasst, die gültig sind.
- Die Nachricht kann nicht mehr als 50 Empfänger in den Feldern „An:“, „CC:“ und „BCC:“ umfassen. Wenn Sie eine E-Mail an eine größere Zielgruppe senden möchten, können Sie Ihre Empfängerliste in Gruppen von höchstens 50 unterteilen und dann die `sendEmail`-Methode aufrufen, um die Nachricht mehrmals an die einzelnen Gruppen zu senden.

Senden von E-Mails

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_sendemail.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, z. B. die Sender- und Empfängeradressen, Betreff, E-Mail-Text im Klartext- und HTML-Format, an die `sendEmail`-Methode der `AWS.SES`-Client-Klasse zu übergeben. Um die `sendEmail`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendemail.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Senden einer E-Mail mit einer Vorlage

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_sendtemplatedemail.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, z. B. die Sender- und Empfängeradressen, Betreff, E-Mail-Text im Klartext- und HTML-Format,

an die `sendTemplatedEmail`-Methode der `AWS.SES`-Client-Klasse zu übergeben. Um die `sendTemplatedEmail`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendtemplatedemail.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Senden einer Massen-E-Mail mithilfe einer Vorlage

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_sendbulktemplatedemail.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, z. B. die Sender- und Empfängeradressen, Betreff, E-Mail-Text im Klartext- und HTML-Format, an die `sendBulkTemplatedEmail`-Methode der `AWS.SES`-Client-Klasse zu übergeben. Um die `sendBulkTemplatedEmail`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verwenden von IP-Adressfiltern für den E-Mail-Empfang in Amazon SES

Dieses Node.js-Codebeispiel zeigt:

- das Erstellen von IP-Adressfiltern, um E-Mails, die von einer IP-Adresse oder einem IP-Adressbereich stammen, zu akzeptieren oder abzulehnen.

- das Auflisten Ihrer aktuellen IP-Adressfilter.
- das Löschen eines IP-Adressfilters.

In Amazon SES ist ein Filter eine Datenstruktur, die aus einem Namen, einem IP-Adressbereich und der Angabe besteht, ob E-Mails aus diesem Bereich zugelassen oder blockiert werden sollen. IP-Adressen, die Sie blockieren oder zulassen möchten, werden als eine einzelne IP-Adresse oder ein IP-Adressbereich in einer Classless Inter-Domain Routing(CIDR)-Notation festgelegt. Einzelheiten darüber, wie Amazon SES E-Mails empfängt, finden Sie unter Amazon SES Email-Receiving Concepts im Amazon Simple Email Service Developer Guide.

Das Szenario

In diesem Beispiel werden mehrere Node.js-Module verwendet, um E-Mails auf verschiedene Weisen zu senden. Die Module Node.js verwenden das SDK für JavaScript die Erstellung und Verwendung von E-Mail-Vorlagen mit den folgenden Methoden der `AWS.SES` Client-Klasse:

- `createReceiptFilter`
- `listReceiptFilters`
- `deleteReceiptFilter`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Konfigurieren des SDKs

Konfigurieren Sie das SDK für, JavaScript indem Sie ein globales Konfigurationsobjekt erstellen und dann die Region für Ihren Code festlegen. In diesem Beispiel ist die Region auf `us-west-2` festgelegt.

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

Erstellen eines IP-Adressfilters

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_createreceiptfilter.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um die Parameterwerte zu übergeben, die den IP-Filter definieren, einschließlich dem Filternamen, einer IP-Adresse bzw. einem bestimmten Adressbereich, nach denen gefiltert werden soll, und ob Datenverkehr von den gefilterten E-Mail-Adressen zugelassen oder blockiert werden soll. Um die `createReceiptFilter`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Der Filter wird in Amazon SES erstellt.

```
node ses_createreceiptfilter.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Auflisten Ihrer IP-Adressfilter

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_listreceiptfilters.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen eines leeren Parameterobjekts. Um die `listReceiptFilters`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

```
// 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);
```

```
  });
```

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES gibt die Filterliste zurück.

```
node ses_listreceiptfilters.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen eines IP-Adressenfilters

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_deletereceiptfilter.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um den Namen des zu löschenden IP-Filters zu übergeben. Um die `deleteReceiptFilter`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Der Filter wird aus Amazon SES gelöscht.

```
node ses_deletereceiptfilter.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verwenden von Empfangsregeln in Amazon SES

Dieses Node.js-Codebeispiel zeigt:

- das Erstellen und Löschen von Empfangsregeln.
- das Organisieren von Empfangsregeln in Gruppen von Empfangsregelsätzen.

Die Empfangsregeln in Amazon SES legen fest, was mit E-Mails geschehen soll, die Sie für E-Mail-Adressen oder Domains erhalten haben, die Sie besitzen. Eine Empfangsregel enthält eine Bedingung und eine sortierte Liste mit Aktionen. Wenn der Empfänger einer eingehenden E-Mail mit einem Empfänger übereinstimmt, der in den Bedingungen für die Empfangsregel angegeben ist, führt Amazon SES die in der Empfangsregel angegebenen Aktionen aus.

Um Amazon SES als E-Mail-Empfänger verwenden zu können, müssen Sie mindestens eine aktive Empfangsregel festgelegt haben. Ein Empfangsregelsatz ist eine geordnete Sammlung von Empfangsregeln, die festlegen, was Amazon SES mit E-Mails tun soll, die Amazon SES über Ihre verifizierten Domains empfängt. Weitere Informationen finden Sie unter Erstellen von Empfangsregeln für den Amazon SES SES-E-Mail-Empfang und Erstellen eines Empfangsregelsatzes für den Amazon SES SES-E-Mail-Empfang im Amazon Simple Email Service Developer Guide.

Das Szenario

In diesem Beispiel werden mehrere Node.js-Module verwendet, um E-Mails auf verschiedene Weisen zu senden. Die Module Node.js verwenden das SDK JavaScript , um E-Mail-Vorlagen mit den folgenden Methoden der `AWS.SES` Client-Klasse zu erstellen und zu verwenden:

- `createReceiptRule`
- `deleteReceiptRule`

- `createReceiptRuleSet`
- `deleteReceiptRuleSet`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer JSON-Datei mit den Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Eine Amazon S3 S3-Empfangsregel erstellen

Jede Empfangsregel für Amazon SES enthält eine geordnete Liste von Aktionen. In diesem Beispiel wird eine Empfangsregel mit einer Amazon S3 S3-Aktion erstellt, die die E-Mail-Nachricht an einen Amazon S3 S3-Bucket zustellt. Einzelheiten zu Aktionen mit Empfangsregeln finden Sie unter Aktionsoptionen im Amazon Simple Email Service Developer Guide.

Damit Amazon SES E-Mails in einen Amazon S3 S3-Bucket schreiben kann, erstellen Sie eine Bucket-Richtlinie, die Amazon SES die `PutObject` Erlaubnis erteilt. Informationen zur Erstellung dieser Bucket-Richtlinie finden Sie unter Erteilen Sie Amazon SES Permission to Write to Your Amazon S3 Bucket im Amazon Simple Email Service Developer Guide.

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine Empfangsregel in Amazon SES zu erstellen, um empfangene Nachrichten in einem Amazon S3 S3-Bucket zu speichern. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_createreceiptrule.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Parameterobjekt, um die erforderlichen Werte zum Erstellen des Empfangsregelsatzes zu übergeben. Um die `createReceiptRuleSet`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES erstellt die Empfangsregel.

```
node ses_createreceiptrule.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen einer Empfangsregel

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_deletereceiptrule.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Parameterobjekt, um den Namen der zu löschenden Empfangsregel zu übergeben. Um die `deleteReceiptRule`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES erstellt die Liste der Empfangsregelsätze.

```
node ses_deletereceiptrule.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Erstellen eines Empfangsregelsatzes

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_createreceiptruleset.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Parameterobjekt, um den Namen des neuen Empfangsregelsatzes zu übergeben. Um die `createReceiptRuleSet`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES erstellt die Liste der Empfangsregelsätze.

```
node ses_createreceiptruleset.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen eines Empfangsregelsatzes

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_deletereceiptruleset.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um den Namen des zu löschenden Empfangsregelsatzes zu übergeben. Um die `deleeReceiptRuleSet`-Methode aufzurufen, erstellen Sie ein Promise für den Aufruf eines Amazon SES-Serviceobjekts und übergeben die Parameter. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein. Amazon SES erstellt die Liste der Empfangsregelsätze.

```
node ses_deletereceiptruleset.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Beispiele für Amazon Simple Notification Service

Amazon Simple Notification Service (Amazon SNS) ist ein Webservice, der die Zustellung oder das Senden von Nachrichten an abonnierende Endpunkte oder Clients koordiniert und verwaltet.

In Amazon SNS gibt es zwei Arten von Kunden — Herausgeber und Abonnenten —, die auch als Produzenten und Verbraucher bezeichnet werden.

Herausgeber kommunizieren asynchron mit Abonnenten, indem sie eine Nachricht erstellen und an ein Thema senden, bei dem es sich wirklich um einen logischen Zugriffspunkt und Kommunikationskanal handelt. Abonnenten (Webserver, E-Mail-Adressen, Amazon SQS-Warteschlangen, Lambda-Funktionen) konsumieren oder empfangen die Nachricht oder Benachrichtigung über eines der unterstützten Protokolle (Amazon SQS, HTTP/S, E-Mail, SMSAWS Lambda), wenn sie das Thema abonniert haben.

Die JavaScript API für Amazon SNS wird über die `Class: AWS.SNS`verfügbar gemacht.

Themen

- Themen in Amazon SNS verwalten
- Nachrichten in Amazon SNS veröffentlichen
- Verwaltung von Abonnements in Amazon SNS
- Senden von SMS-Nachrichten mit Amazon SNS

Themen in Amazon SNS verwalten

Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie Themen in Amazon SNS, zu denen Sie Benachrichtigungen veröffentlichen können.

- So löschen Sie in Amazon SNS erstellte Themen.
- Abrufen einer Liste von verfügbaren Themen
- Abrufen und Festlegen von Themenattributen

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Amazon SNS SNS-Themen zu erstellen, aufzulisten und zu löschen und Themenattribute zu verarbeiten. Die Module Node.js verwenden das SDK JavaScript zur Verwaltung von Themen mithilfe der folgenden Methoden der `AWS.SNS` Client-Klasse:

- `createTopic`
- `listTopics`
- `deleteTopic`
- `getTopicAttributes`
- `setTopicAttributes`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer JSON-Datei mit den Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Erstellen eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Thema zu erstellen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_createtopic.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, um den Namen (`Name`) für das neue Thema an die `createTopic`-Methode der `AWS.SNS`-Client-Klasse zu übergeben. Um die `createTopic` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt

übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback. Die vom Promise-Objekt zurückgegebenen Daten (`data`) enthalten den ARN des Themas.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_createtopic.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Auflisten Ihrer -Themen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um alle Amazon SNS SNS-Themen aufzulisten. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_listtopics.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein leeres Objekt, das an die `listTopics`-Methode der `AWS.SNS`-Client-Klasse übergeben wird. Um die `listTopics` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback. Die vom Promise zurückgegebenen Daten (`data`) enthalten ein Array von ARNs Ihrer Themen.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_listtopics.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Löschen eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Thema zu löschen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_deletetopic.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `TopicArn` des zu löschenden Themas enthält, um es an die `deleteTopic`-Methode der `AWS.SNS`-Client-Klasse zu übergeben. Um die `deleteTopic` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_deletetopic.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Abrufen von Themenattributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um Attribute eines Amazon SNS SNS-Themas abzurufen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_gettopicattributes.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `TopicArn` eines zu löschenden Themas enthält, um es an die `getTopicAttributes`-Methode der `AWS.SNS`-Client-Klasse zu übergeben. Um die `getTopicAttributes` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_gettopicattributes.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Festlegen von Themenattributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die veränderbaren Attribute eines Amazon SNS SNS-Themas festzulegen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_settopicattributes.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das die Parameter für eine Aktualisierung des Attributs enthält. Dazu gehören der `TopicArn` des Themas, dessen Attribute Sie festlegen möchten, der Name des festzulegenden Attributs und der neue Wert für dieses Attribut. Sie können nur die Attribute `Policy`, `DisplayName` und `DeliveryPolicy` festlegen. Übergeben Sie die Parameter an die `setTopicAttributes`-Methode der Client-Klasse `AWS.SNS`. Um die `setTopicAttributes` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_settopicattributes.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Nachrichten in Amazon SNS veröffentlichen

Dieses Node.js-Codebeispiel zeigt:

- So veröffentlichen Sie Nachrichten zu einem Amazon SNS SNS-Thema.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Nachrichten von Amazon SNS an Themenendpunkte, E-Mails oder Telefonnummern zu veröffentlichen. Die Module Node.js verwenden das SDK JavaScript , um Nachrichten mit dieser Methode der `AWS.SNS` Client-Klasse zu senden:

- `publish`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer JSON-Datei mit den Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Eine Nachricht zu einem Amazon SNS SNS-Thema veröffentlichen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine Nachricht zu einem Amazon SNS SNS-Thema zu veröffentlichen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_publishtotopic.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das die Parameter für die Veröffentlichung einer Nachricht enthält, einschließlich des Nachrichtentexts und des ARN des Amazon SNS SNS-Themas. Weitere Details zu verfügbaren SMS-Attributen finden Sie unter SetSMSAttributes.

Übergeben Sie die Parameter an die `publish`-Methode der Client-Klasse `AWS.SNS`. Erstellen Sie ein Versprechen für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die response im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_publishtotopic.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Verwaltung von Abonnements in Amazon SNS

Dieses Node.js-Codebeispiel zeigt:

- So listen Sie alle Abonnements für ein Amazon SNS SNS-Thema auf.
- So abonnieren Sie eine E-Mail-Adresse, einen Anwendungsendpunkt oder eine AWS Lambda Funktion für ein Amazon SNS SNS-Thema.
- So melden Sie sich von Amazon SNS SNS-Themen ab.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Benachrichtigungen zu Amazon SNS SNS-Themen zu veröffentlichen. Die Module Node.js verwenden das SDK JavaScript zur Verwaltung von Themen mithilfe der folgenden Methoden der `AWS.SNS` Client-Klasse:

- `subscribe`
- `confirmSubscription`

- `listSubscriptionsByTopic`
- `unsubscribe`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer JSON-Datei mit den Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Auflisten von Abonnements eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um alle Abonnements für ein Amazon SNS SNS-Thema aufzulisten. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_listsubscriptions.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `TopicArn`-Parameter für das Thema enthält, dessen Abonnements Sie auflisten möchten. Übergeben Sie die Parameter an die `listSubscriptionsByTopic`-Methode der Client-Klasse `AWS.SNS`. Um die `listSubscriptionsByTopic` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_listsubscriptions.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Abonnieren eines Themas durch Hinterlegen einer E-Mail-Adresse

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Adresse zu abonnieren, sodass sie SMTP-E-Mail-Nachrichten von einem Amazon SNS SNS-Thema empfängt. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_subscribeemail.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `Protocol`-Parameter enthält, um das `email`-Protokoll, den `TopicArn` für das Thema, das abonniert werden soll, und eine E-Mail-Adresse als `Endpoint` der Nachricht anzugeben. Übergeben Sie die Parameter an die `subscribe`-Methode der Client-Klasse `AWS.SNS`. Sie können die `subscribe` Methode verwenden, um mehrere verschiedene Endpunkte für ein Amazon SNS SNS-Thema zu abonnieren, abhängig von den Werten, die für die übergebenen Parameter verwendet werden, wie andere Beispiele in diesem Thema zeigen werden.

Um die `subscribe` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_subscribeemail.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Abonnieren eines Themas mit einem Anwendungsendpunkt

Verwenden Sie in diesem Beispiel ein Modul Node.js, um einen mobilen Anwendungsendpunkt zu abonnieren, sodass dieser Benachrichtigungen von einem Amazon SNS SNS-Thema empfängt. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_subscribeapp.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `Protocol`-Parameter enthält, um das `application`-Protokoll, den `TopicArn` für das Thema, das Sie abonnieren möchten, und den ARN des mobilen Anwendungsendpunkts für den `Endpoint`-Parameter anzugeben. Übergeben Sie die Parameter an die `subscribe`-Methode der Client-Klasse `AWS.SNS`.

Um die `subscribe` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_subscribeapp.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Abonnieren einer Lambda-Funktion für ein Thema

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine AWS Lambda Funktion zu abonnieren, sodass sie Benachrichtigungen von einem Amazon SNS SNS-Thema empfängt. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_subscribelambda.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `Protocol`-Parameter enthält, indem Sie das `lambda`-Protokoll, den `TopicArn` für das zu abonnierende Thema und den ARN einer AWS Lambda-Funktion als `Endpoint`-Parameter angeben. Übergeben Sie die Parameter an die `subscribe`-Methode der Client-Klasse `AWS.SNS`.

Um die `subscribe` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_subscribelambda.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Abmelden von einem Thema

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Themenabonnement zu kündigen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_unsubscribe.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `SubscriptionArn`-Parameter enthält, der den ARN des Abonnements angibt, das Sie beenden möchten. Übergeben Sie die Parameter an die `unsubscribe`-Methode der Client-Klasse `AWS.SNS`.

Um die `unsubscribe` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_unsubscribe.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Senden von SMS-Nachrichten mit Amazon SNS

Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie SMS-Nachrichteneinstellungen für Amazon SNS ab und legen sie fest.
- Überprüfen, ob eine Telefonnummer vom Empfang von SMS-Nachrichten abgemeldet wurde
- Abrufen einer Liste der Telefonnummern, die vom Empfang von SMS-Nachrichten abgemeldet wurden
- Senden einer SMS-Nachricht

Das Szenario

Verwenden Sie Amazon SNS, um Textnachrichten oder SMS-Nachrichten an SMS-fähige Geräte zu senden. Sie können eine Nachricht direkt an eine Telefonnummer senden oder Sie können eine Nachricht an mehrere Telefonnummern gleichzeitig senden, indem Sie das Thema für diese Telefonnummern abonnieren und die Nachricht an das Thema senden.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um SMS-Textnachrichten von Amazon SNS auf SMS-fähigen Geräten zu veröffentlichen. Die Module Node.js verwenden das SDK für JavaScript die Veröffentlichung von SMS-Nachrichten mit den folgenden Methoden der `AWS.SNS` Client-Klasse:

- `getSMSAttributes`
- `setSMSAttributes`
- `checkIfPhoneNumberIsOptedOut`
- `listPhoneNumbersOptedOut`
- `publish`

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer JSON-Datei mit den Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Abrufen von SMS-Attributen

Verwenden Sie Amazon SNS, um Einstellungen für SMS-Nachrichten festzulegen, z. B. wie Ihre Lieferungen optimiert werden (aus Kostengründen oder für eine zuverlässige Zustellung), Ihr monatliches Ausgabenlimit, wie Nachrichtenzustellungen protokolliert werden und ob Sie tägliche SMS-Nutzungsberichte abonnieren möchten. Diese Einstellungen werden abgerufen und als SMS-Attribute für Amazon SNS festgelegt.

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die aktuellen SMS-Attribute in Amazon SNS abzurufen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_getsmstype.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein Objekt, das die Parameter zum Abrufen von SMS-Attributen enthält, einschließlich der Namen der einzelnen Attribute, die abgerufen werden. Einzelheiten zu verfügbaren SMS-Attributen finden Sie unter SetSMSAttributes in der Amazon Simple Notification Service API-Referenz.

In diesem Beispiel wird das `DefaultSMSType`-Attribut abgerufen. Dieses Attribut steuert, ob SMS-Nachrichten als `Promotional` oder als `Transactional` gesendet werden. Im ersten Fall wird die Nachrichtenzustellung im Hinblick auf die Kosten und im zweiten Fall im Hinblick auf höchste Zuverlässigkeit optimiert. Übergeben Sie die Parameter an die `setTopicAttributes`-Methode der Client-Klasse `AWS.SNS`. Um die `getSMSAttributes` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_getsmstype.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Festlegen von SMS-Attributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die aktuellen SMS-Attribute in Amazon SNS abzurufen. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_setsmstype.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein Objekt, das die Parameter zum Festlegen von SMS-Attributen enthält, einschließlich der Namen der einzelnen Attribute, die festgelegt werden, und der jeweils festzulegenden Werte. Einzelheiten zu verfügbaren SMS-Attributen finden Sie unter SetSMSAttributes in der Amazon Simple Notification Service API-Referenz.

In diesem Beispiel wird das `DefaultSMSType`-Attribut auf `Transactional` festgelegt. Damit wird die Nachrichtenzustellung im Hinblick auf höchste Zuverlässigkeit optimiert. Übergeben Sie die Parameter an die `setTopicAttributes`-Methode der Client-Klasse `AWS.SNS`. Um die `getSMSAttributes` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_setsmstype.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Überprüfen, ob für eine Telefonnummer der Empfang deaktiviert wurde

In diesem Beispiel verwenden Sie ein Node.js-Modul, um zu überprüfen, ob für eine Telefonnummer der Empfang von SMS-Nachrichten deaktiviert wurde. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_checkphoneoptout.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein Objekt, das die zu überprüfende Telefonnummer als Parameter enthält.

In diesem Beispiel wird der `PhoneNumber`-Parameter festgelegt, um die Telefonnummer anzugeben, die überprüft werden soll. Übergeben Sie das Objekt an die `checkIfPhoneNumberIsOptedOut`-Methode der Client-Klasse `AWS.SNS`. Um die `checkIfPhoneNumberIsOptedOut` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_checkphoneoptout.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Auflisten deaktivierter Telefonnummern

In diesem Beispiel verwenden Sie ein Node.js-Modul, um eine Liste der Telefonnummern abzurufen, für die der Empfang von SMS-Nachrichten deaktiviert wurde. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_listnumbersoptedout.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein leeres Objekt als Parameter.

Übergeben Sie das Objekt an die `listPhoneNumbersOptedOut`-Methode der Client-Klasse `AWS.SNS`. Um die `listPhoneNumbersOptedOut` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_listnumbersoptedout.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Veröffentlichen einer SMS-Nachricht

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden einer SMS-Nachricht an eine Telefonnummer. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sns_publishsms.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein Objekt, das die Parameter `Message` und `PhoneNumber` enthält.

Wenn Sie eine SMS-Nachricht senden, geben Sie die Telefonnummer im E.164-Format an. Die Richtlinie E.164 legt die internationale Schreibweise für Telefonnummern fest. Rufnummern im E.164-Format bestehen aus maximal 15 Zeichen sowie einem vorangestellten Plus-Zeichen (+) und der Ländervorwahl. Eine US-Telefonnummer im E.164-Format sieht beispielsweise wie folgt aus: +1001XXX5550100.

In diesem Beispiel wird der `PhoneNumber`-Parameter festgelegt, um die Telefonnummer zum Senden der Nachricht anzugeben. Übergeben Sie das Objekt an die `publish`-Methode der Client-Klasse `AWS.SNS`. Um die `publish` Methode aufzurufen, erstellen Sie eine Zusage für den Aufruf eines Amazon SNS-Serviceobjekts, indem Sie das Parameterobjekt übergeben. Verarbeiten Sie anschließend die `response` im Promise-Callback.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sns_publishsms.js
```

Diesen Beispielcode finden Sie hier auf. GitHub

Amazon SQS-Beispiele

Amazon Simple Queue Service (Amazon SQS) ist ein schneller, zuverlässiger, skalierbarer, vollständig verwalteter Service für die Nachrichten-Warteschlangen-Service. Mit Amazon SQS können Sie die Komponenten einer Cloud-Anwendung entkoppeln. Amazon SQS umfasst Standardwarteschlangen mit hohem Durchsatz und hoher at-least-once Verarbeitung sowie FIFO-Warteschlangen, die FIFO-Zustellung (First-In, First-Out) und Exactly-Once-Verarbeitung ermöglichen.

Die JavaScript API für Amazon SQS wird über die `AWS.SQS` Client-Klasse verfügbar gemacht. Weitere Informationen zur Verwendung der Amazon SQS SQS-Clientklasse finden Sie `Class: AWS.SQS`in der API-Referenz.

Themen

- Verwenden von Warteschlangen in Amazon SQS
- Senden und Empfangen von Nachrichten in Amazon SQS
- Verwalten der Zeitbeschränkung für die Sichtbarkeit in Amazon SQS
- Aktivieren von Langabfragen in Amazon SQS
- Verwenden von Warteschlangen für unzustellbare Nachrichten in Amazon SQS

Verwenden von Warteschlangen in Amazon SQS

Dieses Node.js-Codebeispiel zeigt:

- Abrufen einer Liste aller Nachrichtenwarteschlangen
- Abrufen der URL für eine bestimmte Warteschlange
- Erstellen und Löschen von Warteschlangen

Informationen zum Beispiel

In diesem Beispiel werden mehrere Node.js-Module für die Arbeit mit Warteschlangen verwendet. Die Module von Node.js verwenden das SDK JavaScript , um Warteschlangen das Aufrufen der folgenden Methoden der `AWS.SQS` Client-Klasse zu ermöglichen:

- `listQueues`
- `createQueue`
- `getQueueUrl`
- `deleteQueue`

Weitere Informationen zu Amazon SQS SQS-Nachrichten finden Sie unter So funktionieren Warteschlangen im Amazon Simple Queue Service Developer Guide.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Auflisten Ihrer Warteschlangen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_listqueues.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon SQS zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Auflisten Ihrer Warteschlangen enthält. Standardmäßig handelt es sich hierbei um ein leeres Objekt. Rufen Sie die `listQueues`-Methode auf, um die Liste der Warteschlangen abzurufen. Die Callback-Funktion gibt die URLs aller Warteschlangen zurück.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_listqueues.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Erstellen einer Warteschlange

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_createqueue.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon SQS zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Auflisten Ihrer Warteschlangen enthält. Diese müssen den Namen für die erstellte Warteschlange beinhalten. Die Parameter können auch Attribute für die Warteschlange enthalten, wie z. B. die Anzahl der Sekunden, um die die Nachrichtenzustellung verzögert wird, oder die Anzahl der Sekunden, für die eine erhaltene Nachricht aufbewahrt wird. Rufen Sie die `createQueue`-Methode auf. Die Callback-Funktion gibt die URL der erstellten Warteschlange zurück.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_createqueue.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Abrufen der URL für eine Warteschlange

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_getqueueurl.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon SQS zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Auflisten Ihrer Warteschlange enthält. Diese müssen den Namen der Warteschlange beinhalten, deren URL Sie abrufen möchten. Rufen Sie die `getQueueUrl`-Methode auf. Die Callback-Funktion gibt die URL der angegebenen Warteschlange zurück.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_getqueueurl.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Löschen einer Warteschlange

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_deletequeue.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon SQS zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Löschen einer Warteschlange enthält. Dieses besteht aus der URL der Warteschlange, die Sie löschen möchten. Rufen Sie die `deleteQueue`-Methode auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_deletequeue.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Senden und Empfangen von Nachrichten in Amazon SQS

Dieses Node.js-Codebeispiel zeigt:

- Senden von Nachrichten in einer Warteschlange
- Empfangen von Nachrichten in einer Warteschlange
- Löschen von Nachrichten in einer Warteschlange

Das Szenario

In diesem Beispiel wird eine Reihe von Node.js-Modulen zum Senden und Empfangen von Nachrichten verwendet. Die Module von Node.js verwenden das SDK JavaScript zum Senden und Empfangen von Nachrichten mithilfe der folgenden Methoden der `AWS.SQS` Client-Klasse:

- `sendMessage`
- `receiveMessage`
- `deleteMessage`

Weitere Informationen zu Amazon SQS SQS-Nachrichten finden Sie unter Senden einer Nachricht an eine Amazon SQS SQS-Warteschlange und Empfangen und Löschen einer Nachricht aus einer Amazon SQS SQS-Warteschlange im Amazon Simple Queue Service Developer Guide.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.

- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen einer Amazon SQS-Warteschlange Ein Beispiel für die Erstellung einer Warteschlange finden Sie unter Verwenden von Warteschlangen in Amazon SQS.

Senden einer Mitteilung an eine Warteschlange

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_sendmessage.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon SQS zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter für Ihre Nachricht enthält, einschließlich URL der Warteschlange, an die Sie die Nachricht senden möchten. In diesem Beispiel enthält die Nachricht Details über ein Buch auf einer Belletristik-Bestsellerliste. Diese umfassen den Titel, den (die) Autor(in) und die Anzahl der Wochen, die es auf der Liste steht.

Rufen Sie die `sendMessage`-Methode auf. Die Callback-Funktion gibt eine eindeutige ID für die Nachricht zurück.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_sendmessage.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Empfangen und Löschen von Nachrichten in einer Warteschlange

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_receivemessage.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon SQS zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter für Ihre Nachricht enthält, einschließlich URL der Warteschlange, von der Sie Nachrichten erhalten möchten. In diesem Beispiel geben die Parameter den Eingang aller Nachrichtenattribute sowie den Eingang von nicht mehr als zehn Nachrichten an.

Rufen Sie die `receiveMessage`-Methode auf. Die Callback-Funktion gibt ein Array von `Message`-Objekten zurück, von dem Sie `ReceiptHandle` für jede verwendete Nachricht abrufen können, um diese Nachricht zu einem späteren Zeitpunkt zu löschen. Erstellen Sie ein weiteres JSON-Objekt, das die erforderlichen Parameter zum Löschen der Nachricht enthält. Dies sind die URL der Warteschlange und der `ReceiptHandle`-Wert. Rufen Sie die `deleteMessage`-Methode auf, um die erhaltene Nachricht zu löschen.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_receivemessage.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verwalten der Zeitbeschränkung für die Sichtbarkeit in Amazon SQS

Dieses Node.js-Codebeispiel zeigt:

- Angeben des Zeitintervalls, in dem Nachrichten, die von einer Warteschlange erhalten wurden, nicht sichtbar sind

Das Szenario

In diesem Beispiel wird ein Node.js-Modul zum Verwalten der Zeitbeschränkung für die Sichtbarkeit verwendet. Das Modul Node.js verwendet das SDK JavaScript , um das Sichtbarkeits-Timeout mithilfe dieser Methode der `AWS.SQS` Client-Klasse zu verwalten:

- `changeMessageVisibility`

Weitere Informationen zum Amazon SQS-Sichtbarkeits-Timeout finden Sie unter Visibility Timeout im Amazon Simple Queue Service Developer Guide.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen einer Amazon SQS-Warteschlange Ein Beispiel für die Erstellung einer Warteschlange finden Sie unter Verwenden von Warteschlangen in Amazon SQS.
- Senden einer Nachricht an die Warteschlange. Ein Beispiel für das Versenden von Nachrichten an eine Warteschlange finden Sie unter Senden und Empfangen von Nachrichten in Amazon SQS.

Ändern der Zeitbeschränkung für die Sichtbarkeit

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_changingvisibility.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon Simple Queue Service zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Empfangen einer Mitteilung aus der Warteschlange

Nachdem Sie die Nachricht aus der Warteschlange erhalten haben, erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter für das Einrichten der Zeitbeschränkung enthält. Dazu gehören die URL der Warteschlange mit der Nachricht, der bei Erhalt der Nachricht zurückgegebene `ReceiptHandle`-Wert sowie die neue Zeitbeschränkung in Sekunden. Rufen Sie die `changeMessageVisibility`-Methode auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_changingvisibility.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Aktivieren von Langabfragen in Amazon SQS

Dieses Node.js-Codebeispiel zeigt:

- Aktivieren von Langabfragen für eine neuerstellte Warteschlange
- Aktivieren von Langabfragen für eine vorhandene Warteschlange
- Aktivieren von Langabfragen bei Nachrichtenempfang

Das Szenario

Lange Abfragen reduzieren die Anzahl leerer Antworten, indem Amazon SQS eine bestimmte Zeit warten kann, bis eine Nachricht in der Warteschlange verfügbar ist, bevor eine Antwort gesendet wird. Durch Langabfragen lassen sich außerdem falsch leere Antworten vermeiden, indem die Anfrage statt an eine Auswahl an Servern an alle Server gesendet wird. Zum Aktivieren von Langabfragen müssen Sie für empfangene Nachrichten eine Wartezeit ungleich Null angeben. Dies können Sie durch Festlegen des `ReceiveMessageWaitTimeSeconds`-Parameters einer

Warteschlange oder des `WaitTimeSeconds`-Parameters für eine Nachricht bei deren Empfang vornehmen.

In diesem Beispiel wird eine Reihe von Node.js-Modulen zum Aktivieren von Langabfragen verwendet. Die Module von Node.js verwenden das SDK JavaScript , um lange Abfragen mithilfe der folgenden Methoden der `AWS.SQS` Client-Klasse zu ermöglichen:

- `setQueueAttributes`
- `receiveMessage`
- `createQueue`

Weitere Informationen zu Amazon SQS Long Polling finden Sie unter Long Polling im Amazon Simple Queue Service Developer Guide.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.

Aktivieren der Langabfrage beim Erstellen einer Warteschlange

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_longpolling_createqueue.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon SQS zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Erstellen einer Warteschlange enthält, einschließlich eines Wertes ungleich Null für den `ReceiveMessageWaitTimeSeconds`-Parameter. Rufen Sie die `createQueue`-Methode auf. Die Langabfrage ist nun für die Warteschlange aktiviert.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_longpolling_createqueue.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Aktivieren der Langabfrage für eine vorhandene Warteschlange

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_longpolling_existingqueue.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon Simple Queue Service zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Festlegen der Warteschlangenattribute enthält. Dazu gehören ein Wert ungleich Null für die `ReceiveMessageWaitTimeSeconds`-Eigenschaft und die URL der Warteschlange. Rufen Sie die `setQueueAttributes`-Methode auf. Die Langabfrage ist nun für die Warteschlange aktiviert.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_longpolling_existingqueue.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Aktivieren von Langabfragen beim Nachrichteneingang

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_longpolling_receivemessage.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon Simple Queue Service zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Empfangen von Nachrichten enthält. Dazu gehören ein Wert ungleich Null für die `WaitTimeSeconds`-Eigenschaft und die URL der Warteschlange. Rufen Sie die `receiveMessage`-Methode auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_longpolling_receivemessage.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Verwenden von Warteschlangen für unzustellbare Nachrichten in Amazon SQS

Dieses Node.js-Codebeispiel zeigt:

- Verwenden einer Warteschlange, um Nachrichten von anderen Warteschlangen zu empfangen und zu halten, die die Warteschlangen nicht verarbeiten können

Das Szenario

Bei einer Warteschlange für unzustellbare Nachrichten handelt es sich um eine Warteschlange, an die andere (Quell-)Warteschlangen Nachrichten senden können, die nicht erfolgreich verarbeitet werden konnten. Sie können diese Nachrichten in der Warteschlange für unzustellbare Nachrichten

sammeln und isolieren, um zu bestimmen, warum die Verarbeitung fehlgeschlagen ist. Sie müssen jede Quellwarteschlange, die Nachrichten an eine Warteschlange für unzustellbare Nachrichten sendet, individuell konfigurieren. Eine Warteschlange für unzustellbare Nachrichten kann von mehreren Warteschlangen verwendet werden.

In diesem Beispiel wird ein Node.js-Modul verwendet, um Nachrichten an eine Warteschlange für unzustellbare Nachrichten weiterzuleiten. Das Modul Node.js verwendet das SDK für JavaScript die Verwendung von Warteschlangen mit unbestätigten Briefen unter Verwendung dieser Methode der `AWS.SQS` Client-Klasse:

- `setQueueAttributes`

Weitere Informationen zu Amazon SQS-Warteschlangen für unzustellbare Briefe finden Sie unter Using Amazon SQS Dead Letter Queues im Amazon Simple Queue Service Developer Guide.

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Installieren Sie Node.js. Weitere Informationen über die Installation von Node.js finden Sie auf der Node.js-Website.
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter Laden der Anmeldeinformationen in Node.js aus der freigegebenen Anmeldeinformationsdatei.
- Erstellen Sie eine Amazon SQS SQS-Warteschlange, die als Warteschlange für unzustellbare Nachrichten dient. Ein Beispiel für die Erstellung einer Warteschlange finden Sie unter Verwenden von Warteschlangen in Amazon SQS.

Konfigurieren von Quellwarteschlangen

Nachdem Sie eine Warteschlange für unzustellbare Nachrichten erstellt haben, müssen Sie die anderen Warteschlangen konfigurieren, damit diese unverarbeiteten Nachrichten an die Warteschlange für unzustellbare Nachrichten weiterleiten. Geben Sie hierfür eine Redrive-Richtlinie an, die die als Warteschlange für unzustellbare Nachrichten zu verwendende Warteschlange identifiziert und legen Sie die maximale Anzahl der Empfangsversuche für einzelne Nachrichten fest, bevor sie an die Warteschlange für unzustellbare Nachrichten weitergeleitet werden.

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `sqs_deadletterqueue.js`. Stellen Sie sicher, dass Sie das SDK, wie zuvor dargestellt, konfigurieren. Um auf Amazon SQS zuzugreifen, erstellen Sie ein `AWS.SQS` Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die erforderlichen Parameter zum Aktualisieren von Warteschlangenattributen enthält, einschließlich des `RedrivePolicy`-Parameters, der sowohl den ARN der Warteschlange für unzustellbare Nachrichten als auch den Wert von `maxReceiveCount` angibt. Geben Sie auch die URL der Quellwarteschlange an, die Sie konfigurieren möchten. Rufen Sie die `setQueueAttributes`-Methode auf.

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

Um das Beispiel auszuführen, geben Sie Folgendes in der Befehlszeile ein.

```
node sqs_deadletterqueue.js
```

Diesen Beispielcode finden Sie hier auf GitHub.

Tutorials

Die folgenden Tutorials veranschaulichen, wie Sie verschiedene Aufgaben im Zusammenhang mit dem AWS SDK for JavaScript durchführen.

Themen

Tutorial: Node.js auf einer Amazon EC2 EC2-Instance einrichten

Ein gängiges Szenario für die Verwendung von Node.js mit dem SDK für JavaScript ist die Einrichtung und Ausführung einer Node.js -Webanwendung auf einer Amazon Elastic Compute Cloud (Amazon EC2) -Instance. In diesem Tutorial erstellen Sie eine Linux-Instance, stellen eine Verbindung zur Instance über SSH her und installieren anschließend Node.js, um es auf dieser Instance auszuführen.

Voraussetzungen

In diesem Tutorial wird davon ausgegangen, dass Sie bereits eine Linux-Instance mit einem öffentlichen DNS-Namen gestartet haben, die über das Internet erreichbar ist und mit der Sie eine Verbindung über SSH herstellen können. Weitere Informationen finden Sie unter Schritt 1: Starten einer Instance im Amazon EC2 EC2-Benutzerhandbuch.

> **Important**
>
> Verwenden Sie das Amazon Linux 2023 Amazon Machine Image (AMI), wenn Sie eine neue Amazon EC2 EC2-Instance starten.

Außerdem müssen Sie Ihre Sicherheitsgruppe so konfiguriert haben, dass Verbindungen über SSH (Port 22), HTTP (Port 80) und HTTPS (Port 443) erlaubt sind. Weitere Informationen zu diesen Voraussetzungen finden Sie unter Setting Up with Amazon Amazon EC2 im Amazon EC2 EC2-Benutzerhandbuch.

Verfahren

Mithilfe des folgenden Verfahrens können Sie Node.js auf einer Amazon Linux-Instance installieren. Sie können diesen Server zum Hosten einer Node.js-Webanwendung verwenden.

So richten Sie Node.js auf Ihrer Linux-Instance ein

1. Stellen Sie als `ec2-user` eine Verbindung mit Ihrer Linux-Instance über SSH her.
2. Installieren Sie den Node-Versionsmanager (nvm). Geben Sie dazu den folgenden Befehl in die Befehlszeile ein.

   > **Warning**
   >
   > AWS kontrolliert den folgenden Code nicht. Bevor Sie ihn ausführen, überprüfen Sie unbedingt dessen Authentizität und Integrität. Weitere Informationen zu diesem Code finden Sie im GitHub nvm-Repository.

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
   ```

   Wir verwenden nvm zum Installieren von Node.js, da nvm mehrere Versionen von Node.js installieren kann und die Möglichkeit bietet, zwischen diesen zu wechseln.
3. Laden Sie, `nvm` indem Sie in der Befehlszeile Folgendes eingeben.

   ```
   source ~/.bashrc
   ```
4. Verwenden Sie nvm, um die neueste LTS-Version von Node.js zu installieren, indem Sie in der Befehlszeile Folgendes eingeben.

   ```
   nvm install --lts
   ```

   Bei der Installation von Node.js wird auch der Node Package Manager (npm) installiert, sodass Sie bei Bedarf zusätzliche Module installieren können.
5. Testen Sie, ob Node.js installiert ist und ordnungsgemäß ausgeführt wird. Geben Sie dazu den folgenden Befehl in die Befehlszeile ein.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

Dadurch erscheint folgende Meldung, in der die ausgeführte Node.js-Version anzeigt wird.

```
Running Node.js VERSION
```

> **Note**
>
> Die Knoteninstallation gilt nur für die aktuelle Amazon EC2-Sitzung. Wenn Sie Ihre CLI-Sitzung neu starten, müssen Sie nvm verwenden, um die installierte Knotenversion zu aktivieren. Wenn die Instanz beendet ist, müssen Sie den Knoten erneut installieren. Die Alternative besteht darin, ein Amazon Machine Image (AMI) der Amazon EC2 EC2-Instance zu erstellen, sobald Sie die Konfiguration haben, die Sie behalten möchten, wie im folgenden Thema beschrieben.

Erstellen eines Amazon Machine Image (AMI)

Nachdem Sie Node.js auf einer Amazon EC2 EC2-Instance installiert haben, können Sie aus dieser Instance ein Amazon Machine Image (AMI) erstellen. Die Erstellung eines AMI macht es einfach, mehrere Amazon EC2 EC2-Instances mit derselben Node.js Installation bereitzustellen. Weitere Informationen zum Erstellen eines AMI aus einer vorhandenen Instance finden Sie unter Creating an Amazon EBS-backed Linux AMI im Amazon EC2 EC2-Benutzerhandbuch.

Verwandte Ressourcen

Weitere Informationen über die in diesem Thema verwendete(n) Befehle und Software finden Sie auf den folgenden Webseiten:

- Node Version Manager (nvm): siehe nvm repo on. GitHub
- Node Packet Manager (npm): npm Website

JavaScript API-Referenz

Die API-Referenzthemen für die neueste Version des SDK für JavaScript finden Sie unter:

APIAWS SDK for JavaScript -Referenzhandbuch .

SDK-Änderungsprotokoll auf GitHub

Das Änderungsprotokoll für Veröffentlichungen von Version 2.4.8 und höher finden Sie unter:

Änderungsprotokoll .

Sicherheit für dieses AWS Produkt oder diese Dienstleistung

Cloud-Sicherheit genießt bei Amazon Web Services (AWS) höchste Priorität. Als AWS -Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die zur Erfüllung der Anforderungen von Organisationen entwickelt wurden, für die Sicherheit eine kritische Bedeutung hat. Sicherheit ist eine gemeinsame Verantwortung zwischen Ihnen AWS und Ihnen. Im Modell der übergreifenden Verantwortlichkeit wird Folgendes mit „Sicherheit der Cloud“ bzw. „Sicherheit in der Cloud“ umschrieben:

Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der alle in der AWS Cloud angebotenen Dienste ausgeführt werden, und für die Bereitstellung von Diensten, die Sie sicher nutzen können. Unsere Sicherheitsverantwortung hat bei uns höchste Priorität AWS, und die Wirksamkeit unserer Sicherheit wird im Rahmen der AWS Compliance-Programme regelmäßig von externen Prüfern getestet und verifiziert.

Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem von Ihnen genutzten AWS Dienst und anderen Faktoren, wie der Sensibilität Ihrer Daten, den Anforderungen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften.

Dieses AWS Produkt oder dieser Service folgt dem Modell der gemeinsamen Verantwortung in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der Seite mit der Dokumentation zur AWS Servicesicherheit und den AWS Services, für die das AWS Compliance-Programm zur Einhaltung der Vorschriften zuständig ist.

Themen

Datenschutz in diesem AWS Produkt oder dieser Dienstleistung

Das AWS Modell der gilt für den Datenschutz in diesem AWS Produkt oder dieser Dienstleistung. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS -Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS -Modell der geteilten Verantwortung und in der DSGVO im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Verwenden Sie SSL/TLS, um mit Ressourcen zu kommunizieren. AWS Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein. AWS CloudTrail
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS -Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-2.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit diesem AWS Produkt oder Service oder einem anderen AWS -Services über die Konsole, API oder SDKs arbeiten. AWS CLI AWS Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine

Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Identitäts- und Zugriffsverwaltung

AWS Identity and Access Management (IAM) hilft einem Administrator AWS -Service , den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS IAM ist ein Programm AWS -Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

Zielgruppe

Die Art und Weise, wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der Arbeit ab, in der Sie tätig sind. AWS

Dienstbenutzer — Wenn Sie dies AWS -Services für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen zur Verfügung. Wenn Sie für Ihre Arbeit mehr AWS Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Fuktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anzufordern müssen. Falls Sie auf eine Funktion in der von Ihnen verwendeten Version nicht zugreifen können AWS, finden Fehlerbehebung bei AWS Identität und Zugriff Sie weitere Informationen in der Bedienungsanleitung des AWS -Service Geräts, das Sie verwenden.

Serviceadministrator — Wenn Sie in Ihrem Unternehmen für die AWS Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf AWS. Es ist Ihre Aufgabe, zu bestimmen, auf welche AWS Funktionen und Ressourcen Ihre Servicebenutzer zugreifen sollen. Sie müssen dann Anträge an Ihren IAM-Administrator stellen, um die Berechtigungen Ihrer Servicenutzer zu ändern.

Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen. Weitere Informationen darüber, wie Ihr Unternehmen IAM verwenden kann AWS, finden Sie in der Benutzeranleitung des von AWS -Service Ihnen verwendeten.

IAM-Administrator: Wenn Sie als IAM-Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf AWS verfassen können. Beispiele für AWS identitätsbasierte Richtlinien, die Sie in IAM verwenden können, finden Sie im Benutzerhandbuch der AWS -Service von Ihnen verwendeten.

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen als IAM-Benutzer authentifiziert (angemeldet AWS) sein oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich AWS als föderierte Identität anmelden, indem Sie Anmeldeinformationen verwenden, die über eine Identitätsquelle bereitgestellt wurden. AWS IAM Identity Center (IAM Identity Center) -Benutzer, die Single Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für föderierte Identitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie über den Verbund darauf zugreifen AWS , übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich beim AWS Management Console oder beim AWS Zugangsportal anmelden. Weitere Informationen zur Anmeldung finden Sie AWS unter So melden Sie sich bei Ihrem an AWS-Konto im AWS-Anmeldung Benutzerhandbuch.

Wenn Sie AWS programmgesteuert zugreifen, AWS stellt es ein Software Development Kit (SDK) und eine Befehlszeilenschnittstelle (CLI) bereit, um Ihre Anfragen mithilfe Ihrer Anmeldeinformationen kryptografisch zu signieren. Wenn Sie keine AWS Tools verwenden, müssen Sie Anfragen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode, um Anfragen selbst zu signieren, finden Sie im IAM-Benutzerhandbuch unter AWS API-Anfragen signieren.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. AWS Empfiehlt beispielsweise, die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter Multi-Faktor-Authentifizierung im AWS IAM Identity Center -Benutzerhandbuch und Verwenden der Multi-Faktor-Authentifizierung (MFA) in AWS im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS -Services Ressourcen im Konto hat. Diese Identität wird als AWS-Konto Root-Benutzer bezeichnet. Sie können darauf zugreifen, indem Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen und verwenden Sie diese, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

Verbundidentität

Als bewährte Methode sollten menschliche Benutzer, einschließlich Benutzer, die Administratorzugriff benötigen, für den Zugriff AWS -Services mithilfe temporärer Anmeldeinformationen den Verbund mit einem Identitätsanbieter verwenden.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensbenutzerverzeichnis, einem Web-Identitätsanbieter AWS Directory Service, dem Identity Center-Verzeichnis oder einem beliebigen Benutzer, der mithilfe AWS -Services von Anmeldeinformationen zugreift, die über eine Identitätsquelle bereitgestellt wurden. Wenn föderierte Identitäten darauf zugreifen AWS-Konten, übernehmen sie Rollen, und die Rollen stellen temporäre Anmeldeinformationen bereit.

Für die zentrale Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center zu verwenden. Sie können Benutzer und Gruppen in IAM Identity Center erstellen, oder Sie können eine Verbindung zu einer Gruppe von Benutzern und Gruppen in Ihrer eigenen Identitätsquelle herstellen und diese synchronisieren, um sie in all Ihren AWS-Konten Anwendungen zu verwenden. Informationen zu IAM Identity Center finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center - Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über spezifische Berechtigungen für eine einzelne Person oder Anwendung verfügt. Wenn möglich, empfehlen wir, temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter Regelmäßiges

Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

Eine IAM-Gruppe ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise einer Gruppe mit dem Namen IAMAdmins Berechtigungen zum Verwalten von IAM-Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter Erstellen eines IAM-Benutzers (anstatt einer Rolle) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über bestimmte Berechtigungen verfügt. Sie ist einem IAM-Benutzer vergleichbar, ist aber nicht mit einer bestimmten Person verknüpft. Sie können vorübergehend eine IAM-Rolle in der übernehmen, AWS Management Console indem Sie die Rollen wechseln. Sie können eine Rolle übernehmen, indem Sie eine AWS CLI oder AWS API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter Verwenden von IAM-Rollen im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- Verbundbenutzerzugriff – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter Erstellen von Rollen für externe Identitätsanbieter im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter Berechtigungssätze im AWS IAM Identity Center -Benutzerhandbuch.
- Temporäre IAM-Benutzerberechtigungen – Ein IAM-Benutzer oder eine -Rolle kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.

- Kontoübergreifender Zugriff – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können Sie AWS -Services jedoch eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zum Unterschied zwischen Rollen und ressourcenbasierten Richtlinien für den kontenübergreifenden Zugriff finden Sie unter Kontenübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.
- Serviceübergreifender Zugriff — Einige verwenden Funktionen in anderen. AWS -Services AWS -Services Wenn Sie beispielsweise einen Aufruf in einem Service tätigen, führt dieser Service häufig Anwendungen in Amazon-EC2 aus oder speichert Objekte in Amazon-S3. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
  - Forward Access Sessions (FAS) — Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service initiieren. FAS verwendet die Berechtigungen des Prinzipals, der einen aufruft AWS -Service, in Kombination mit der Anfrage, Anfragen an AWS -Service nachgelagerte Dienste zu stellen. FAS-Anfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS -Services oder Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.
  - Servicerolle – Eine Servicerolle ist eine IAM-Rolle, die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS -Service im IAM-Benutzerhandbuch.
  - Dienstbezogene Rolle — Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS -Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Servicebezogene Rollen erscheinen in Ihrem Dienst AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.
- Anwendungen, die auf Amazon EC2 ausgeführt werden — Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2-Instance ausgeführt werden und API-Anfragen stellen AWS CLI . AWS Das ist eher zu empfehlen, als Zugriffsschlüssel innerhalb der EC2-Instance zu speichern. Um einer EC2-Instance eine AWS Rolle zuzuweisen und sie allen ihren Anwendungen zur Verfügung zu stellen, erstellen Sie

ein Instance-Profil, das an die Instance angehängt ist. Ein Instance-Profil enthält die Rolle und ermöglicht, dass Programme, die in der EC2-Instance ausgeführt werden, temporäre Anmeldeinformationen erhalten. Weitere Informationen finden Sie unter Verwenden einer IAM-Rolle zum Erteilen von Berechtigungen für Anwendungen, die auf Amazon-EC2-Instances ausgeführt werden im IAM-Benutzerhandbuch.

Informationen dazu, wann Sie IAM-Rollen oder IAM-Benutzer verwenden sollten, finden Sie unter Erstellen einer IAM-Rolle (anstatt eines Benutzers) im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie ist ein Objekt, AWS das, wenn es einer Identität oder Ressource zugeordnet ist, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anfrage stellt. Berechtigungen in den Richtlinien bestimmen, ob die Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen von der AWS Management Console AWS CLI, der oder der AWS API abrufen.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern,

welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Erstellen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie mehreren Benutzern, Gruppen und Rollen in Ihrem System zuordnen können AWS-Konto. Zu den verwalteten Richtlinien gehören AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer eingebundenen Richtlinie wählen, finden Sie unter Auswahl zwischen verwalteten und eingebundenen Richtlinien im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS -Services

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffssteuerungslisten (ACLs)

Zugriffssteuerungslisten (ACLs) steuern, welche Prinzipale (Kontomitglieder, Benutzer oder Rollen) auf eine Ressource zugreifen können. ACLs sind ähnlich wie ressourcenbasierte Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Services, die ACLs unterstützen. AWS WAF Weitere Informationen“ zu ACLs finden Sie unter Zugriffskontrollliste (ACL) – Übersicht (Access Control List) im Amazon-Simple-Storage-Service-Entwicklerhandbuch.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger verbreitete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- Berechtigungsgrenzen – Eine Berechtigungsgrenze ist ein erweitertes Feature, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (IAM-Benutzer oder -Rolle) erteilen kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen über Berechtigungsgrenzen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im IAM-Benutzerhandbuch.
- Service Control Policies (SCPs) — SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) in festlegen. AWS Organizations AWS Organizations ist ein Dienst zur Gruppierung und zentralen Verwaltung mehrerer Objekte AWS-Konten , die Ihrem Unternehmen gehören. Wenn Sie innerhalb einer Organisation alle Features aktivieren, können Sie Service-Kontrollrichtlinien (SCPs) auf alle oder einzelne Ihrer Konten anwenden. Das SCP schränkt die Berechtigungen für Entitäten in Mitgliedskonten ein, einschließlich der einzelnen Entitäten. Root-Benutzer des AWS-Kontos Weitere Informationen zu Organizations und SCPs finden Sie unter Funktionsweise von SCPs im AWS Organizations -Benutzerhandbuch.
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen darüber, wie AWS bestimmt wird,

ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie im IAM-Benutzerhandbuch unter Bewertungslogik für Richtlinien.

Wie AWS -Services arbeiten Sie mit IAM

Einen allgemeinen Überblick darüber, wie die meisten IAM-Funktionen AWS -Services funktionieren, finden Sie im AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren.

Informationen zur Verwendung bestimmter Dienste AWS -Service mit IAM finden Sie im Abschnitt Sicherheit im Benutzerhandbuch des jeweiligen Dienstes.

Fehlerbehebung bei AWS Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS und IAM auftreten können.

Themen

- Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS
- Ich bin nicht berechtigt, iam durchzuführen: PassRole
- Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS Ressourcen ermöglichen

Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, über die Konsole Details zu einer fiktiven *`my-example-widget`*-Ressource anzuzeigen, jedoch nicht über `awes:`*`GetWidget`*-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der `awes:`*`GetWidget`*-Aktion auf die *`my-example-widget`*-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam durchzuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWSübergeben zu können.

Einige AWS -Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS Ressourcen ermöglichen

Sie können eine Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf Ihre Ressourcen verwenden können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Im Fall von Diensten, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (Access Control Lists, ACLs) verwenden, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob diese Funktionen AWS unterstützt werden, finden Sie unterWie AWS - Services arbeiten Sie mit IAM.
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.

- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie im IAM-Benutzerhandbuch unter Kontenübergreifender Ressourcenzugriff in IAM.

Überprüfung der Einhaltung der Vorschriften für dieses Produkt oder diese Dienstleistung AWS

Informationen darüber, ob AWS -Service ein AWS -Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS -Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS -Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- Schnellstartanleitungen zu Sicherheit und Compliance — In diesen Bereitstellungsleitfäden werden architektonische Überlegungen erörtert und Schritte für die Bereitstellung von Basisumgebungen beschrieben AWS , bei denen Sicherheit und Compliance im Mittelpunkt stehen.
- Architecting for HIPAA Security and Compliance on Amazon Web Services — In diesem Whitepaper wird beschrieben, wie Unternehmen HIPAA-fähige Anwendungen erstellen AWS können.

> **Note**
>
> AWS -Services Nicht alle sind HIPAA-fähig. Weitere Informationen finden Sie in der Referenz für HIPAA-berechtigte Services.

- AWS Compliance-Ressourcen — Diese Sammlung von Arbeitsmappen und Leitfäden gilt möglicherweise für Ihre Branche und Ihren Standort.
- AWS Leitfäden zur Einhaltung von Vorschriften für Kunden — Verstehen Sie das Modell der gemeinsamen Verantwortung aus dem Blickwinkel der Einhaltung von Vorschriften. In den Leitfäden werden die bewährten Verfahren zur Sicherung zusammengefasst AWS -Services und die Leitlinien den Sicherheitskontrollen in verschiedenen Frameworks (einschließlich des National Institute of Standards and Technology (NIST), des Payment Card Industry Security Standards Council (PCI) und der International Organization for Standardization (ISO)) zugeordnet.
- Evaluierung von Ressourcen anhand von Regeln im AWS Config Entwicklerhandbuch — Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub— Auf diese AWS -Service Weise erhalten Sie einen umfassenden Überblick über Ihren internen Sicherheitsstatus. AWS Security Hub verwendet Sicherheitskontrollen, um Ihre AWS -Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten Methoden zu überprüfen. Eine Liste der unterstützten Services und Kontrollen finden Sie in der Security-Hub-Steuerungsreferenz.
- Amazon GuardDuty — Dies AWS -Service erkennt potenzielle Bedrohungen für Ihre Workloads AWS-Konten, Container und Daten, indem es Ihre Umgebung auf verdächtige und böswillige Aktivitäten überwacht. GuardDuty kann Ihnen helfen, verschiedene Compliance-Anforderungen wie PCI DSS zu erfüllen, indem es die in bestimmten Compliance-Frameworks vorgeschriebenen Anforderungen zur Erkennung von Eindringlingen erfüllt.
- AWS Audit Manager— Auf diese AWS -Service Weise können Sie Ihre AWS Nutzung kontinuierlich überprüfen, um das Risikomanagement und die Einhaltung von Vorschriften und Industriestandards zu vereinfachen.

Dieses AWS Produkt oder dieser Service folgt dem Modell der gemeinsamen Verantwortung in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der Seite mit der Dokumentation zur AWS Servicesicherheit und den AWS Services, für die das AWS Compliance-Programm zur Einhaltung der Vorschriften zuständig ist.

Ausfallsicherheit für dieses AWS Produkt oder diese Dienstleistung

Die AWS globale Infrastruktur basiert auf AWS-Regionen Availability Zones.

AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind.

Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Dieses AWS Produkt oder dieser Service folgt dem Modell der gemeinsamen Verantwortung in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der Seite mit der Dokumentation zur AWS Servicesicherheit und den AWS Services, für die das AWS Compliance-Programm zur Einhaltung der Vorschriften zuständig ist.

Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service

Dieses AWS Produkt oder dieser Dienst verwendet Managed Services und ist daher durch die AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter AWS Cloud-Sicherheit. Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf dieses AWS Produkt oder diesen Service zuzugreifen. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit AWS

Security Token Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Dieses AWS Produkt oder dieser Service folgt dem Modell der gemeinsamen Verantwortung in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der Seite mit der Dokumentation zur AWS Servicesicherheit und den AWS Services, für die das AWS Compliance-Programm zur Einhaltung der Vorschriften zuständig ist.

Erzwingen einer Mindestversion von TLS

> **Important**
>
> Die Version AWS SDK for JavaScript v2 handelt automatisch die höchste TLS-Version aus, die von einem bestimmten AWS Service-Endpunkt unterstützt wird. Sie können optional eine TLS-Mindestversion erzwingen, die für Ihre Anwendung erforderlich ist, z. B. TLS 1.2 oder 1.3. Beachten Sie jedoch, dass TLS 1.3 von einigen AWS Service-Endpunkten nicht unterstützt wird, sodass einige Aufrufe fehlschlagen können, wenn Sie TLS 1.3 erzwingen.

Um die Sicherheit bei der Kommunikation mit AWS Diensten zu erhöhen, konfigurieren Sie den AWS SDK for JavaScript für die Verwendung von TLS 1.2 oder höher.

Transport Layer Security (TLS) ist ein Protokoll, das von Webbrowsern und anderen Anwendungen verwendet wird, um die Privatsphäre und Integrität der über ein Netzwerk ausgetauschten Daten zu gewährleisten.

Überprüfen und Erzwingen von TLS in Node.js

Wenn Sie das AWS SDK for JavaScript mit Node.js verwenden, wird die zugrunde liegende Sicherheitsebene Node.js verwendet, um die TLS-Version festzulegen.

Node.js 12.0.0 und höher verwenden eine Mindestversion von OpenSSL 1.1.1b, die TLS 1.3 unterstützt. AWS SDK for JavaScript Version 3 verwendet standardmäßig TLS 1.3, sofern verfügbar, verwendet jedoch standardmäßig eine niedrigere Version, falls erforderlich.

Überprüfen der Version von OpenSSL und TLS

Führen Sie den folgenden Befehl aus, um die von Node.js verwendete Version von OpenSSL auf Ihrem Computer abzurufen.

```
node -p process.versions
```

Die Version von OpenSSL in der Liste ist die von Node.js verwendete Version, wie im folgenden Beispiel gezeigt.

```
openssl: '1.1.1b'
```

Um die von Node.js verwendete Version von TLS auf Ihrem Computer abzurufen, starten Sie die Knoten-Shell und führen Sie die folgenden Befehle in der angegebenen Reihenfolge aus.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

Der letzte Befehl gibt die TLS-Version aus, wie im folgenden Beispiel gezeigt.

```
'TLSv1.3'
```

Node.js verwendet standardmäßig diese Version von TLS und versucht, eine andere Version von TLS auszuhandeln, wenn ein Aufruf nicht erfolgreich ist.

Erzwingen einer Mindestversion von TLS

Node.js verhandelt eine Version von TLS, wenn ein Aufruf fehlschlägt. Sie können während dieser Verhandlung die zulässige Mindestversion von TLS durchsetzen, entweder wenn Sie ein Skript über die Befehlszeile ausführen oder per Anfrage in Ihrem Code. JavaScript

Um die minimale TLS-Version über die Befehlszeile anzugeben, müssen Sie Node.js Version 11.0.0 oder höher verwenden. Um eine bestimmte Node.js-Version zu installieren, installieren Sie zuerst Node Version Manager (nvm) mit den Schritten unter Node Version Manager Installing and Updating. Führen Sie dann die folgenden Befehle aus, um eine bestimmte Version von Node.js zu installieren und zu verwenden.

```
nvm install 11
nvm use 11
```

Enforcing TLS 1.2

Um zu erzwingen, dass TLS 1.2 die minimal zulässige Version ist, geben Sie beim Ausführen des Skripts das Argument `--tls-min-v1.2` an, wie im folgenden Beispiel gezeigt.

```
node --tls-min-v1.2 yourScript.js
```

Um die zulässige TLS-Mindestversion für eine bestimmte Anfrage in Ihrem JavaScript Code anzugeben, verwenden Sie den `httpOptions` Parameter, um das Protokoll anzugeben, wie im folgenden Beispiel gezeigt.

```
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

Um zu erzwingen, dass TLS 1.3 die zulässige Mindestversion ist, geben Sie das `--tls-min-v1.3` Argument bei der Ausführung Ihres Skripts an, wie im folgenden Beispiel gezeigt.

```
node --tls-min-v1.3 yourScript.js
```

Um die zulässige TLS-Mindestversion für eine bestimmte Anfrage in Ihrem JavaScript Code anzugeben, verwenden Sie den `httpOptions` Parameter, um das Protokoll anzugeben, wie im folgenden Beispiel gezeigt.

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

Überprüfen und Erzwingen von TLS in einem Browserskript

Wenn Sie das SDK für JavaScript in einem Browserskript verwenden, steuern die Browsereinstellungen die verwendete TLS-Version. Die vom Browser verwendete Version von TLS kann nicht vom Skript erkannt oder festgelegt werden und muss vom Benutzer konfiguriert werden. Informationen zur Überprüfung und Erzwingung der in einem Browserskript verwendeten Version von TLS finden Sie in den Anweisungen für den jeweiligen Browser.

Microsoft Internet Explorer

1. Öffnen Sie Internet Explorer.
2. Wählen Sie in der Menüleiste die Option Extras — Internetoptionen — Registerkarte Erweitert.
3. Scrollen Sie nach unten zur Kategorie Sicherheit und aktivieren Sie manuell das Optionsfeld für TLS 1.2 verwenden.
4. Klicken Sie auf OK.
5. Schließen Sie Ihren Browser und starten Sie Internet Explorer neu.

Microsoft Edge

1. Geben Sie im Suchfeld des Windows-Menüs *Internetoptionen* ein.
2. Klicken Sie unter Beste Übereinstimmung auf Internetoptionen.
3. Scrollen Sie im Fenster Interneteigenschaften auf der Registerkarte Erweitert nach unten zum Abschnitt Sicherheit.
4. Markieren Sie das Kontrollkästchen User TLS 1.2.

5. Klicken Sie auf OK.

Google Chrome

1. Öffnen Sie Google Chrome.
2. Klicken Sie auf Alt F und wählen Sie Einstellungen.
3. Scrollen Sie nach unten und wählen Sie Erweiterte Einstellungen anzeigen... .
4. Scrollen Sie nach unten zum Abschnitt System und klicken Sie auf Proxyeinstellungen öffnen... .
5. Wählen Sie die Registerkarte Erweitert aus.
6. Scrollen Sie nach unten zur Kategorie Sicherheit und aktivieren Sie manuell das Optionsfeld für TLS 1.2 verwenden.
7. Klicken Sie auf OK.
8. Schließen Sie Ihren Browser und starten Sie Google Chrome neu.

Mozilla Firefox

1. Öffnen Sie Firefox.
2. Geben Sie in der Adressleiste about:config ein und drücken Sie die Eingabetaste.
3. Geben Sie im Suchfeld tls ein. Suchen Sie den Eintrag für security.tls.version.min und doppelklicken Sie darauf.
4. Setzen Sie den Integer-Wert auf 3, um zu erzwingen, dass das Protokoll von TLS 1.2 als Standard verwendet wird.
5. Klicken Sie auf OK.
6. Schließen Sie Ihren Browser und starten Sie Mozilla Firefox neu.

Apple Safari

Es gibt keine Optionen zum Aktivieren von SSL-Protokollen. Wenn Sie Safari Version 7 oder höher verwenden, wird TLS 1.2 automatisch aktiviert.

Weitere Ressourcen

Die folgenden Links bieten zusätzliche Ressourcen, die Sie mit dem AWS SDK for JavaScript verwenden können.

AWSSDKs- und Tools-Referenz

DieAWSSDKs- und Tools-Referenzenthält auch Einstellungen, Funktionen und andere grundlegende Konzepte, die in vielen derAWSSDKs.

JavaScript SDK-Forum

Fragen und Diskussionen zu Themen, die für Anwender des SDK von Interesse sind, finden Sie unter JavaScript imJavaScript SDK-Forum.

JavaScript SDK- und Entwicklerhandbuch auf GitHub

Es gibt mehrere Repositorys auf GitHub für das SDK für JavaScript.

- Das aktuelle SDK für JavaScript ist in der Region verfügbarSDK-Repo.
- Das SDK für JavaScript -Entwicklerhandbuch (dieses Dokument) finden Sie im Markdown-Format in seinem eigenen FormatDokumentations-Repo.
- Ein Teil des Beispielcodes in diesem Handbuch ist im SDK-Beispielcode-Repository verfügbar.

JavaScript SDK auf Gitter

Fragen und Diskussionen über das SDK finden Sie auch für JavaScript imJavaScript SDK-Communityauf Gitter.

Dokumentverlauf für AWS SDK for JavaScript

- SDK-Version: Aufrufen JavaScript API-Referenz
- Letzte wichtige Aktualisierung der Dokumentation: 31. März 2022

Dokumentverlauf

In der folgenden Tabelle werden wichtige Änderungen in den einzelnen Versionen des AWS SDK for JavaScript beschrieben, die nach Mai 2018 veröffentlicht wurden. Für Benachrichtigungen über Aktualisierungen dieser Dokumentation können Sie einen RSS-Feed abonnieren.

Änderung	Beschreibung	Datum
Durchsetzung einer Mindestversion von TLS	Es wurden Informationen zu TLS 1.3 hinzugefügt.	31. März 2022
Fotos in einem Amazon S3 S3-Bucket von einem Browser aus anzeigen	Ein Beispiel für einfaches Anzeigen von Fotos in vorhandenen Fotoalben hinzugefügt.	13. Mai 2019
Anmeldeinformationen in Node.js einrichten, neue Optionen zum Laden von Anmeldeinformationen	Weitere Informationen zu Anmeldeinformationen hinzugefügt, die von dem ECS Anmeldeinformationsanbieter oder über einen konfiguri erten Prozess für Anmeldein formationen geladen werden.	25. April 2019
Anmeldeinformationen mithilfe eines konfigurierten Anmeldein formationsprozesses	Weitere Informationen zu Anmeldeinformationen, die über den Prozess für die Konfigurierung von Anmeldein formationen geladen werden	25. April 2019
Neu Erste Schritte in einem Browser-Skript	Getting Started in a Browser Script wurde neu geschrieben,	14. Juli 2018

	um das Beispiel zu vereinfachen und den Zugriff auf den Amazon Polly-Service zu ermöglichen, um Text zu senden und synthetisierte Sprache zurückzugeben, die Sie im Browser abspielen können. Weitere Informationen finden Sie unter Erste Schritte im Browser-Skript für den neuen Inhalt.	
Neue Amazon SNS SNS-Codebeispiele	Vier neue Node.js Codebeispiele für die Arbeit mit Amazon SNS wurden hinzugefügt. Den Beispielcode finden Sie in den Amazon SNS SNS-Beispielen.	29. Juni 2018
Neue Erste Schritte in Node.js	Erste Schritte in Node.js wurde umgeschrieben, um aktualisierten Beispielcode zu verwenden und mehr Details zum Erstellen der `package.json` -Datei sowie des Node.js-Codes selbst bereitzustellen. Weitere Informationen finden Sie unter Erste Schritte in Node.js für den neuen Inhalt.	4. Juni 2018

Frühere Aktualisierungen

In der folgenden Tabelle werden die wichtigen Änderungen in den einzelnen Versionen des AWS SDK for JavaScript vor Juni 2018 beschrieben.

Änderung	Beschreibung	Datum
Neue AWS Elemental MediaConvert-Codebeispiele	Drei neue Node.js-Codebeispiele für die Arbeit mit AWS Elemental MediaConvert wurden hinzugefügt. Einen Beispiel-Code finden Sie unter Beispiele für AWS Elemental MediaConvert.	21. Mai 2018
Neue GitHub Schaltfläche „Auf bearbeiten“	In der Kopfzeile jedes Themas befindet sich jetzt eine Schaltfläche, über die Sie zur Markdown-Version desselben Themas gelangen, GitHub sodass Sie Änderungen vornehmen können, um die Genauigkeit und Vollständigkeit des Leitfadens zu verbessern.	21. Februar 2018
Neues Thema zu benutzerdefinierten Endpunkten	Informationen zum Format und der Verwendung von benutzerdefinierten Endpunkten für das Ausrichten von API-Aufrufen wurden hinzugefügt. Siehe Festlegen von benutzerdefinierten Endpunkten.	20. Februar 2018
Leitfaden zum SDK für JavaScript Entwickler unter GitHub	Das SDK for JavaScript Developer Guide ist im Markdown-Format in einem eigenen Dokumentationsrepo verfügbar. Sie können Probleme veröffentlichen, die das Handbuch angehen	16. Februar 2018

Änderung	Beschreibung	Datum
	soll, oder senden Sie Pull-Anforderungen, um vorgeschlagenen Änderungen zu übermitteln.	
Neues Amazon DynamoDB DynamoDB-Codebeispiel	Ein neues Node.js Codebeispiel für die Aktualisierung einer DynamoDB-Tabelle mit dem Document Client wurde hinzugefügt. Einen Beispiel-Code finden Sie unter Verwenden des DynamoDB-Dokumentenclients.	14. Februar 2018
Neues Thema zu AWS Cloud9	Ein Thema zur Verwendung von AWS Cloud9 für die Entwicklung und das Debuggen von Browser- und Node.js-Code wurde hinzugefügt. Siehe Verwenden von AWS Cloud9 mit der AWS SDK for JavaScript.	5. Februar 2018
Neues Thema zur SDK-Protokollierung	Es wurde ein Thema hinzugefügt, das beschreibt, wie API-Aufrufe protokolliert werden, die mit dem SDK für getätigt wurden. Es JavaScript enthält auch Informationen zur Verwendung eines Loggers eines Drittanbieters. Siehe Protokollieren von AWS SDK for JavaScript-Aufrufen.	5. Februar 2018

Änderung	Beschreibung	Datum
Aktualisiertes Thema zur Einstellung der Region	Das Thema zur Festlegung der Region, die mit dem SDK verwendet wird, wurde aktualisiert und erweitert, einschließlich Informationen über die Rangfolge bei der Einstellung der Region. Siehe Einstellung der AWS Region.	12. Dezember 2017
Neue Amazon SES SES-Codebeispiele	Der Abschnitt mit SDK-Codebeispielen wurde aktualisiert und enthält nun fünf neue Beispiele für die Arbeit mit Amazon SES. Weitere Informationen zu diesen Codebeispielen finden Sie unter Beispiele für Amazon Simple Email Service.	9. November 2017

Änderung	Beschreibung	Datum
Verbesserung der Benutzerfreundlichkeit	Basierend auf den jüngsten Nutzungstests wurde eine Reihe von Änderungen vorgenommen, um die Benutzerfreundlichkeit der Dokumentation zu verbessern. • Codebeispiele werden genauer als für Browser- oder Node.js-Ausführungen ausgerichtet identifiziert. • TOC-Links springen nicht mehr sofort zu anderen Webinhalten, einschließlich der API-Referenz. • Enthält im Abschnitt Erste Schritte weitere Links zu Einzelheiten zum Abrufen von AWS Anmeldeinformationen. • Stellt mehr Informationen zu gemeinsamen Node.js-Funktionen bereit, die zur Verwendung des SDKs benötigt werden. Weitere Informationen finden Sie unter Überlegungen zu Node.js.	9. August 2017

Änderung	Beschreibung	Datum
Neue DynamoDB-Codebeispiele	Der Abschnitt mit SDK-Codebeispielen wurde aktualisiert, um die beiden vorherigen Beispiele neu zu schreiben und drei brandneue Beispiele für die Arbeit mit DynamoDB hinzuzufügen. Weitere Informationen zu diesen Codebeispielen finden Sie unter Amazon DynamoDB-Beispiele.	21. Juni 2017
Neue IAM-Codebeispiele	Der Abschnitt mit SDK-Codebeispielen wurde aktualisiert und enthält nun fünf neue Beispiele für die Arbeit mit IAM. Weitere Informationen zu diesen Codebeispielen finden Sie unter AWSIAM-Beispiele.	23. Dezember 2016
Neue CloudWatch Codebeispiele und Amazon SQS SQS-Codebeispiele	Der Abschnitt mit SDK-Codebeispielen wurde aktualisiert und enthält nun neue Beispiele für die Arbeit mit CloudWatch und mit Amazon SQS. Weitere Informationen zu diesen Codebeispielen finden Sie unter CloudWatch Amazon-Beispiele und Amazon SQS-Beispiele.	20. Dezember 2016

Änderung	Beschreibung	Datum
Neue Amazon EC2 EC2-Codebeispiele	Der Abschnitt mit SDK-Codebeispielen wurde aktualisiert und enthält nun fünf neue Beispiele für die Arbeit mit Amazon EC2. Weitere Informationen zu diesen Codebeispielen finden Sie unter Amazon EC2-Beispiele.	15. Dezember 2016
Liste der unterstützten Browser sichtbarer gemacht	Die Liste der vom SDK für unterstützten Browser JavaScript, die zuvor im Thema Voraussetzungen zu finden war, wurde mit einem eigenen Thema versehen, um sie im Inhaltsverzeichnis besser sichtbar zu machen.	16. November 2016
Erstveröffentlichung des neuen Entwicklerhandbuchs	Das vorherige Entwicklerhandbuch ist jetzt veraltet. Das neue Entwicklerhandbuch wurde neu organisiert, um Informationen leichter auffindbar zu machen. Wenn entweder Node.js oder JavaScript Browserszenarien besondere Überlegungen beinhalten, werden diese als angemessen gekennzeichnet. Das Handbuch bietet außerdem zusätzliche Codebeispiele, die besser organisiert sind, damit sie einfacher und schneller gefunden werden können.	28. Oktober 2016