Guida per gli sviluppatori di SDK v2

AWS SDK per JavaScript

AWS SDK per JavaScript: Guida per gli sviluppatori di SDK v2

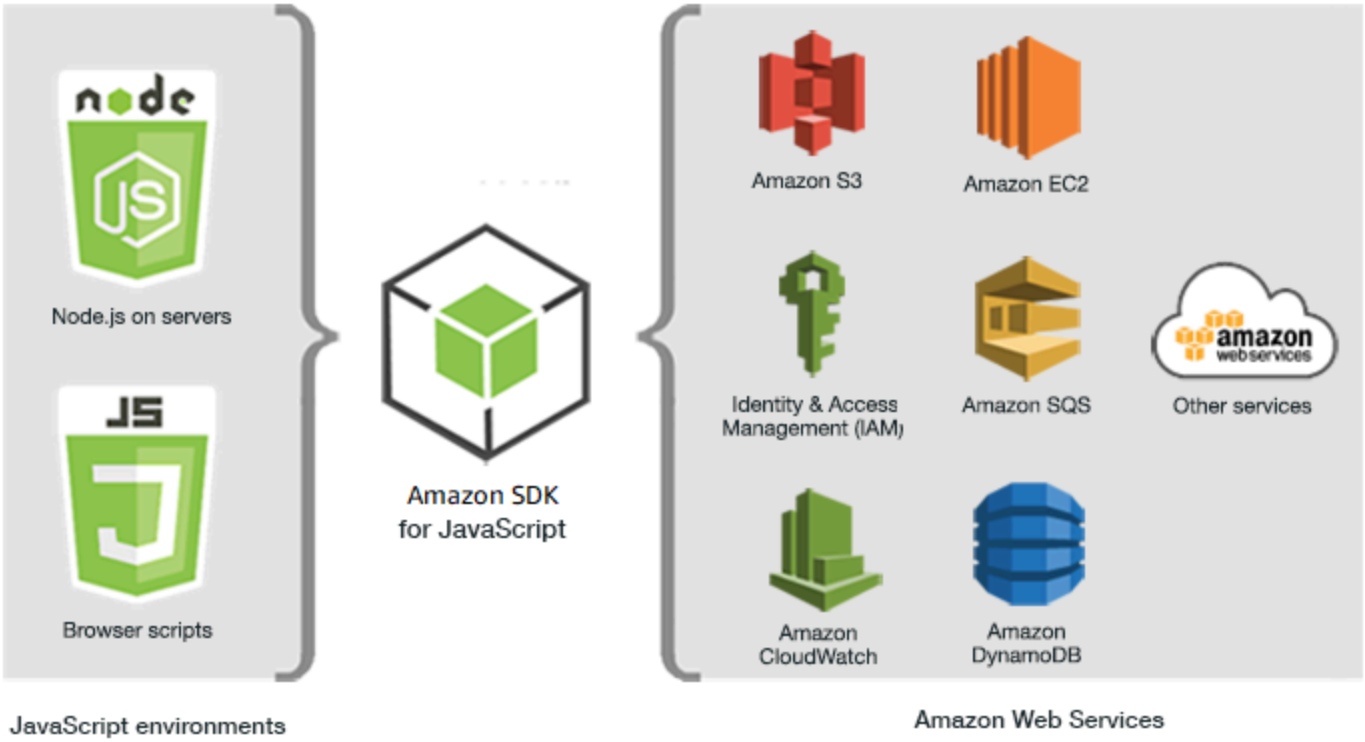
Table of Contents

La AWS SDK per JavaScript v2 è arrivata. end-of-support Ti consigliamo di migrare alla AWS SDK per JavaScript v3. Per ulteriori dettagli e informazioni su come effettuare la migrazione, consulta questo annuncio.

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.

Che cos'è il AWS SDK per JavaScript?

AWS SDK per JavaScriptFornisce un' JavaScript API per i AWS servizi. È possibile utilizzare l' JavaScript API per creare librerie o applicazioni per Node.js o per il browser.



Non tutti i servizi sono immediatamente disponibili nell'SDK. Per scoprire quali servizi sono attualmente supportati da AWS SDK per JavaScript, consulta https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md. Per informazioni sull'SDK for JavaScript on GitHub, consulta. Risorse aggiuntive

Manutenzione e supporto per le versioni principali dell'SDK

Per informazioni sulla manutenzione e il supporto per le versioni principali dell'SDK e le relative dipendenze sottostanti, consulta quanto segue nella AWS SDKs and Tools Reference Guide:

- AWS SDKs e politica di manutenzione degli strumenti
- AWS SDKs e matrice di supporto delle versioni degli strumenti

Utilizzo dell'SDK con Node.js

Node.js è un runtime multipiattaforma per l'esecuzione di applicazioni lato server JavaScript . Puoi configurare Node.js su un' EC2 istanza Amazon per l'esecuzione su un server. Puoi anche usare Node.js per scrivere AWS Lambda funzioni su richiesta.

L'utilizzo dell'SDK per Node.js è diverso dal modo in cui lo si utilizza JavaScript in un browser Web. La differenza sta nel modo in cui si carica l'SDK e in cui si ottengono le credenziali necessarie per accedere a servizi Web specifici. Quando l'uso di particolari APIs differisce tra Node.js e il browser, tali differenze verranno evidenziate.

Usare l'SDK con Amplify AWS

Per le app web, mobili e ibride basate su browser, puoi anche utilizzare AWS Amplify Library on GitHub, che estende l'SDK per fornire un'interfaccia dichiarativa. JavaScript

> **Note**
>
> Framework come AWS Amplify potrebbero non offrire lo stesso supporto per i browser dell'SDK. JavaScript Controlla la documentazione di ciascun framework per i dettagli.

Utilizzo dell'SDK con i browser Web

Tutti i principali browser Web supportano l'esecuzione di. JavaScript JavaScriptil codice in esecuzione in un browser Web viene spesso definito lato client JavaScript.

L'utilizzo dell'SDK for JavaScript in un browser Web è diverso dal modo in cui lo si utilizza per Node.js. La differenza sta nel modo in cui si carica l'SDK e in cui si ottengono le credenziali necessarie per accedere a servizi Web specifici. Quando l'uso di particular è APIs diverso tra Node.js e il browser, tali differenze verranno evidenziate.

Per un elenco dei browser supportati da AWS SDK per JavaScript, vedereBrowser Web supportati.

Casi di utilizzo comune

L'utilizzo dell'SDK per JavaScript gli script del browser consente di realizzare una serie di casi d'uso interessanti. Di seguito sono riportate diverse idee su cosa è possibile creare in un'applicazione browser utilizzando l'SDK per accedere JavaScript a vari servizi Web.

- Crea una console personalizzata per AWS i servizi in cui puoi accedere e combinare funzionalità di diverse regioni e servizi per soddisfare al meglio le tue esigenze organizzative o di progetto.
- Usa Amazon Cognito Identity per consentire l'accesso degli utenti autenticati alle applicazioni del browser e ai siti Web, incluso l'uso dell'autenticazione di terze parti da Facebook e altri.
- Usa Amazon Kinesis per elaborare flussi di clic o altri dati di marketing in tempo reale.
- Usa Amazon DynamoDB per la persistenza dei dati senza server, ad esempio le preferenze dei singoli utenti per i visitatori del sito Web o gli utenti delle applicazioni.
- AWS Lambda Usalo per incapsulare la logica proprietaria che puoi richiamare dagli script del browser senza scaricare e rivelare la tua proprietà intellettuale agli utenti.

Informazioni sugli esempi

Puoi consultare l'SDK per trovare esempi nella Code Example Library JavaScript .AWS

Guida introduttiva alla AWS SDK per JavaScript

AWS SDK per JavaScript Fornisce l'accesso ai servizi Web negli script del browser o in Node.js. Questa sezione contiene due esercizi introduttivi che mostrano come utilizzare l'SDK for JavaScript in ciascuno di questi JavaScript ambienti.

Argomenti

Nozioni di base su uno script di browser

Questo script di browser di esempio illustra:

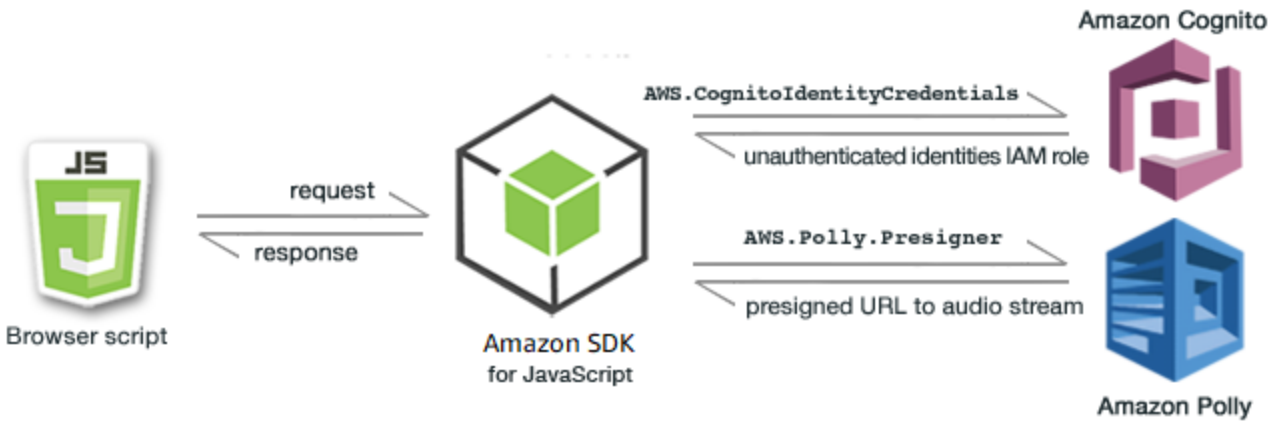
- Come accedere ai AWS servizi da uno script del browser utilizzando Amazon Cognito Identity.
- Come trasformare il testo in sintesi vocale con Amazon Polly.
- Come utilizzare un oggetto presigner per creare un URL prefirmato.

Lo scenario

Amazon Polly è un servizio cloud che converte il testo in voce naturale. Puoi utilizzare Amazon Polly per sviluppare applicazioni che aumentano il coinvolgimento e l'accessibilità. Amazon Polly supporta più lingue e include una varietà di voci realistiche. Per ulteriori informazioni su Amazon Polly, consulta la Amazon Polly Developer Guide.

L'esempio mostra come configurare ed eseguire un semplice script del browser che prende il testo immesso, lo invia ad Amazon Polly e quindi restituisce l'URL dell'audio sintetizzato del testo da riprodurre. Lo script del browser utilizza Amazon Cognito Identity per fornire le credenziali necessarie per accedere ai servizi. AWS Vedrai gli schemi di base per il caricamento e l'utilizzo dell'SDK per JavaScript gli script del browser.

> **Note**
>
> La riproduzione della sintesi vocale in questo esempio dipende dall'esecuzione in un browser che supporta HTML 5 audio.



Lo script del browser utilizza l'SDK per JavaScript sintetizzare il testo utilizzando questi: APIs

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

Fase 1: creare un pool di identità di Amazon Cognito

In questo esercizio, crei e utilizzi un pool di identità Amazon Cognito per fornire un accesso non autenticato allo script del browser per il servizio Amazon Polly. La creazione di un pool di identità crea anche due ruoli IAM, uno per supportare gli utenti autenticati da un provider di identità e l'altro per supportare gli utenti guest non autenticati.

In questo esercizio, lavoreremo solo con il ruolo degli utenti non autenticati in modo che l'attività risulti mirata. È possibile integrare il supporto per un provider di identità e per gli utenti autenticati in un secondo momento. Per ulteriori informazioni sull'aggiunta di un pool di identità di Amazon Cognito, consulta il Tutorial: Creazione di un pool di identità nella Amazon Cognito Developer Guide.

Per creare un pool di identità Amazon Cognito

1. Accedi Console di gestione AWS e apri la console Amazon Cognito all'indirizzo. https://console.aws.amazon.com/cognito/
2. Nel riquadro di navigazione a sinistra, scegli Identity pool.
3. Scegli Crea pool di identità.
4. In Configura la fiducia del pool di identità, scegli Accesso ospite per l'autenticazione degli utenti.
5. In Configura le autorizzazioni, scegli Crea un nuovo ruolo IAM e inserisci un nome (ad esempio, getStartedRole) nel nome del ruolo IAM.
6. In Configure properties, inserisci un nome (ad esempio, getStartedPool) in Identity pool name.
7. In Esamina e crea, conferma le selezioni effettuate per il nuovo pool di identità. Seleziona Modifica per tornare alla procedura guidata e modificare le eventuali impostazioni. Al termine, seleziona Crea un pool di identità.
8. Prendi nota dell'ID del pool di identità e della regione del pool di identità di Amazon Cognito appena creato. Questi valori sono necessari per sostituirli *IDENTITY_POOL_ID* e *REGION* inserirli. Fase 4: scrivere lo script di browser

Dopo aver creato il tuo pool di identità Amazon Cognito, sei pronto per aggiungere le autorizzazioni per Amazon Polly necessarie allo script del tuo browser.

Fase 2: aggiungere una policy al ruolo IAM creato

Per abilitare l'accesso tramite script del browser ad Amazon Polly per la sintesi vocale, utilizza il ruolo IAM non autenticato creato per il tuo pool di identità Amazon Cognito. Ciò richiede l'aggiunta di una policy IAM al ruolo. Per ulteriori informazioni sulla modifica dei ruoli IAM, consulta Modifica della politica di autorizzazione di un ruolo nella Guida per l'utente IAM.

Per aggiungere una policy Amazon Polly al ruolo IAM associato a utenti non autenticati

1. Accedi Console di gestione AWS e apri la console IAM all'indirizzo. https://console.aws.amazon.com/iam/
2. Nel pannello di navigazione a sinistra, seleziona Ruoli.
3. Scegli il nome del ruolo che desideri modificare (ad esempio, getStartedRole), quindi scegli la scheda Autorizzazioni.
4. Seleziona Aggiungi autorizzazioni, quindi seleziona Collega policy.

5. Nella pagina Aggiungi autorizzazioni per questo ruolo, trova e seleziona la casella di controllo per. AmazonPollyReadOnly

> **Note**
>
> Puoi utilizzare questo processo per abilitare l'accesso a qualsiasi AWS servizio.

6. Scegli Add Permissions (Aggiungi autorizzazioni).

Dopo aver creato il tuo pool di identità Amazon Cognito e aggiunto le autorizzazioni per Amazon Polly al tuo ruolo IAM per utenti non autenticati, sei pronto per creare la pagina Web e lo script del browser.

Fase 3: creare la pagina HTML

L'applicazione di esempio è composta da un'unica pagina HTML che contiene l'interfaccia utente e lo script di browser. Per iniziare, è necessario creare un documento HTML e copiare i seguenti contenuti al suo interno. La pagina include un campo e un pulsante di input, un elemento `<audio>` per riprodurre la sintesi vocale e un elemento <p> per visualizzare i messaggi. (L'intero esempio viene visualizzato nella parte inferiore di questa pagina.)

Per ulteriori informazioni sull'elemento `<audio>`, consulta 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>
```

Salva il file HTML, denominandolo `polly.html`. Dopo aver creato l'interfaccia utente per l'applicazione, è possibile aggiungere il codice dello script di browser che esegue l'applicazione.

Fase 4: scrivere lo script di browser

La prima cosa da fare quando si crea lo script del browser è includere l'SDK per JavaScript aggiungendo un `<script>` elemento dopo l'elemento nella pagina. `<audio>` Per trovare l'SDK_VERSION_NUMBER corrente, consulta l'API Reference for the SDK consultabile nella API Reference Guide. JavaScript AWS SDK per JavaScript

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

Quindi aggiungere un nuovo elemento `<script type="text/javascript">` dopo la voce SDK. Lo script di browser sarà aggiunto a questo elemento. Imposta la regione e le credenziali per l'SDK. AWS Quindi, crea una funzione denominata `speakText()` che verrà richiamata come gestore eventi tramite il pulsante.

Per sintetizzare il parlato con Amazon Polly, devi fornire una serie di parametri tra cui il formato audio dell'output, la frequenza di campionamento, l'ID della voce da utilizzare e il testo da riprodurre. Quando crei i parametri, imposta il parametro `Text:` su una stringa vuota; il parametro `Text:` verrà impostato sul valore recuperato dall'elemento `<input>` nella pagina Web. Sostituisci *IDENTITY_POOL_ID* e *REGION* nel codice seguente con i valori indicati in. Fase 1: creare un pool di identità di Amazon Cognito

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

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

        // Function invoked by button click
        function speakText() {
            // Create the JSON parameters for getSynthesizeSpeechUrl
            var speechParams = {
                OutputFormat: "mp3",
                SampleRate: "16000",
```

```
            Text: "",
            TextType: "text",
            VoiceId: "Matthew"
        };
        speechParams.Text = document.getElementById("textEntry").value;
```

Amazon Polly restituisce il parlato sintetizzato come flusso audio. Il modo più semplice per riprodurre l'audio in un browser è fare in modo che Amazon Polly renda l'audio disponibile a un URL predefinito che puoi quindi impostare come `src` attributo dell'`<audio>`elemento nella pagina Web.

Crea un nuovo oggetto di servizio `AWS.Polly`. Quindi crea l'oggetto `AWS.Polly.Presigner` che verrà utilizzato per creare l'URL prefirmato da cui l'audio della sintesi vocale può essere recuperato. È necessario passare i parametri della sintesi definiti, nonché l'oggetto di servizio `AWS.Polly` creato al costruttore `AWS.Polly.Presigner`.

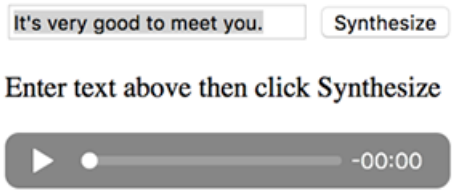
Una volta creato l'oggetto presigner, chiama il metodo `getSynthesizeSpeechUrl` di tale oggetto, passando i parametri della sintesi. In caso di successo, questo metodo restituisce l'URL della sintesi vocale, che è possibile assegnare all'elemento `<audio>` per la riproduzione.

```
        // 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>
```

Fase 5: eseguire l'esempio

Per eseguire l'applicazione di esempio, carica `polly.html` in un browser Web. La presentazione del browser dovrebbe avere un aspetto simile.



Inserisci una frase che desideri venga sintetizzata nella casella di input, quindi scegli Synthesize (Sintetizza). Quando l'audio è pronto per la riproduzione, viene visualizzato un messaggio. Utilizza i controlli del lettore audio per ascoltare la sintesi vocale.

Esempio completo

Questa è la pagina HTML completa con lo script di browser. È disponibile anche qui. 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>
```

Possibili miglioramenti

Ecco alcune varianti di questa applicazione che puoi utilizzare per esplorare ulteriormente l'utilizzo dell'SDK per JavaScript uno script del browser.

- Esercitarsi con altri formati di output audio.
- Aggiungi l'opzione per selezionare una delle varie voci fornite da Amazon Polly.
- Integra un provider di identità come Facebook o Amazon da utilizzare con il ruolo IAM autenticato.

Nozioni di base su Node.js

Questo esempio di codice di Node.js illustra:

- Come creare il manifest `package.json` per il progetto.
- Come installare e includere i moduli utilizzati dal progetto.
- Come creare un oggetto di servizio Amazon Simple Storage Service (Amazon S3) dalla `AWS.S3` classe client.
- Come creare un bucket Amazon S3 e caricare un oggetto in quel bucket.

Lo scenario

L'esempio mostra come configurare ed eseguire un semplice modulo Node.js che crea un bucket Amazon S3, quindi vi aggiunge un oggetto di testo.

Poiché i nomi dei bucket in Amazon S3 devono essere univoci a livello globale, questo esempio include un modulo Node.js di terze parti che genera un valore ID univoco che puoi incorporare nel nome del bucket. Questo modulo aggiuntivo è denominato `uuid`.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Creazione di una directory di lavoro per sviluppare il modulo Node.js. Denomina questa directory `awsnodesample`. La directory deve essere creata in una posizione in cui le applicazioni possano aggiornarla. In Windows, ad esempio, la directory non va creata in "C:\Program Files".
- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js. È possibile trovare i download delle versioni attuali e LTS di Node.js per una varietà di sistemi operativi all'indirizzo https://nodejs. org/en/download/current/.

Indice

Passaggio 1: installa l'SDK e le dipendenze

Si installa l'SDK per il JavaScript pacchetto utilizzando npm (il gestore di pacchetti Node.js).

Dalla directory `awsnodesample` nel pacchetto, digitare quanto segue nella riga di comando.

```
npm install aws-sdk
```

Questo comando installa l'SDK for JavaScript nel progetto e lo aggiorna per `package.json` elencare l'SDK come dipendenza del progetto. Per trovare informazioni su questo pacchetto, cercare "aws-sdk" nel sito Web npm.

Quindi, installare il modulo `uuid` per il progetto digitando quanto segue nella riga di comando, che installa il modulo e aggiorna `package.json`. Per maggiori informazioni in meritouuid, consulta la pagina del modulo all'indirizzo uuid. https://www.npmjs.com/package/

```
npm install uuid
```

Questi pacchetti e il relativo codice vengono installati nella sottodirectory `node_modules` del progetto.

Per ulteriori informazioni sull'installazione dei pacchetti Node.js, consulta la sezione su come scaricare e installare i pacchetti in locale e su come creare moduli Node.js nel sito Web npm (Node.js package manager). Per informazioni sul download e l'installazione di AWS SDK per JavaScript, vedere. Installazione dell'SDK per JavaScript

Fase 2: Configurazione delle credenziali

Devi fornire le credenziali AWS in modo che l'SDK acceda solo al tuo account e alle relative risorse. Per ulteriori informazioni su come ottenere le credenziali dell'account, consulta Autenticazione SDK con AWS.

Per conservare queste informazioni, consigliamo di creare un file di credenziali condiviso. Per scoprire come, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso. I file delle credenziali devono essere simili al seguente esempio.

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

È possibile determinare se le credenziali sono state impostate correttamente eseguendo il codice seguente con Node.js:

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

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

Analogamente, se la regione è stata impostata correttamente nel `config` file, è possibile visualizzare tale valore impostando la variabile di `AWS_SDK_LOAD_CONFIG` ambiente su qualsiasi valore e utilizzando il codice seguente:

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

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

Fase 3: Creare il Package JSON per il progetto

Una volta creata la directory del progetto `awsnodesample`, è necessario creare e aggiungere un file `package.json` per memorizzare i metadati per il progetto Node.js. Per i dettagli sull'utilizzo `package.json` in un progetto Node.js, consulta Creazione di un file package.json.

Nella directory del progetto, creare un nuovo file denominato `package.json`. Aggiungere il JSON al file.

```
{
  "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"
}
```

Salvare il file. Dopo aver installato i moduli necessari, la porzione `dependencies` del file sarà completata. Puoi trovare un file JSON che mostra un esempio di queste dipendenze qui. GitHub

Fase 4: scrivere il codice Node.js

Creare un nuovo file denominato `sample.js` che contiene il codice di esempio. Inizia aggiungendo le chiamate di `require` funzione per includere l'SDK JavaScript e `uuid` i moduli in modo che siano disponibili all'uso.

Crea un nome di bucket univoco da utilizzare per creare un bucket Amazon S3 aggiungendo un valore ID univoco a un prefisso riconoscibile, in questo caso. `'node-sdk-sample-'` È possibile generare l'ID univoco chiamando il modulo `uuid`. Quindi creare un nome per il parametro `Key` utilizzato per caricare un oggetto sul bucket.

Creare un oggetto `promise` per chiamare il metodo `createBucket` dell'oggetto di servizio `AWS.S3`. Come risposta in caso di esito positivo, creare i parametri necessari per caricare il testo sul nuovo bucket. Utilizzando un'altra promessa, chiamare il metodo `putObject` per caricare l'oggetto di testo sul bucket.

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

Questo codice di esempio può essere trovato qui. GitHub

Fase 5: eseguire l'esempio

Digitare il comando seguente per eseguire l'esempio.

```
node sample.js
```

Se il caricamento viene completato correttamente, appare un messaggio di conferma nella riga di comando. È inoltre possibile trovare il bucket e l'oggetto di testo caricato nella console Amazon S3.

Configurazione dell'SDK per JavaScript

Gli argomenti di questa sezione spiegano come installare l'SDK JavaScript per utilizzarlo nei browser Web e con Node.js. Mostra inoltre come caricare l'SDK per poter accedere ai servizi Web supportati dall'SDK.

> **Note**
>
> Gli sviluppatori di React Native dovrebbero AWS Amplify usare per creare nuovi progetti su AWS. Consulta l'aws-sdk-react-nativearchivio per i dettagli.

Argomenti

Prerequisiti

Prima di utilizzare il AWS SDK per JavaScript, stabilite se il codice deve essere eseguito in Node.js o nei browser Web. Successivamente, esegui queste operazioni:

- Per Node.js, installa Node.js sui tuoi server, se non è già installato.
- Per i browser Web, identifica le versioni del browser che è necessario supportare.

Argomenti

Configurazione di un ambiente AWS Node.js

Per configurare un ambiente AWS Node.js in cui eseguire l'applicazione, utilizzate uno dei seguenti metodi:

- Scegli un'Amazon Machine Image (AMI) con Node.js preinstallato e crea un'istanza Amazon EC2 utilizzando quell'AMI. Quando crei la tua istanza Amazon EC2, scegli il tuo AMI tra. Marketplace AWS Marketplace AWS Cerca Node.js e scegli un'opzione AMI che includa una versione di Node.js (32 o 64 bit) preinstallata.
- Crea un'istanza Amazon EC2 e installa Node.js su di essa. Per ulteriori informazioni su come installare Node.js su un'istanza Amazon Linux, consultaTutorial: configurazione di Node.js su un'istanza Amazon EC2.
- Crea un ambiente serverless utilizzando AWS Lambda to run Node.js come funzione Lambda. Per ulteriori informazioni sull'utilizzo di Node.js all'interno di una funzione Lambda, consulta Programming Model (Node.js) nella AWS Lambda Developer Guide.
- Distribuisci la tua applicazione Node.js su. AWS Elastic BeanstalkPer ulteriori informazioni sull'utilizzo di Node.js con Elastic Beanstalk, consulta Deploying Node.js AWS Elastic Beanstalk Applications to nella Developer Guide.AWS Elastic Beanstalk

Browser Web supportati

L'SDK for JavaScript supporta tutti i browser Web moderni, incluse queste versioni minime:

Browser	Versione
Google Chrome	28.0+
Mozilla Firefox	26.0+
Opera	17.0+
Microsoft Edge	25.10+
Windows Internet Explorer	N/D
Apple Safari	5+
Browser Android	4.3+

> **Note**
>
> I framework come AWS Amplify potrebbero non offrire lo stesso supporto per i browser dell'SDK. JavaScript Controlla la documentazione di ciascun framework per i dettagli.

Installazione dell'SDK per JavaScript

Se e come installarla AWS SDK per JavaScript dipende dal fatto che il codice venga eseguito nei moduli Node.js o negli script del browser.

Non tutti i servizi sono immediatamente disponibili nell'SDK. Per scoprire quali servizi sono attualmente supportati da AWS SDK per JavaScript, vedere https://github.com/aws/aws-sdk-js/.md blob/master/SERVICES

Node

Il modo preferito per installare AWS SDK per JavaScript for Node.js consiste nell'utilizzare npm, il gestore di pacchetti Node.js. A tale scopo, digita quanto segue nella riga di comando.

```
npm install aws-sdk
```

Se viene visualizzato questo messaggio di errore:

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

Digita i seguenti comandi nella riga di comando:

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

Browser

Non è necessario installare l'SDK per utilizzarlo negli script del browser. Puoi caricare il pacchetto SDK ospitato direttamente da Amazon Web Services con uno script nelle tue pagine HTML. Il pacchetto SDK ospitato supporta il sottoinsieme di AWS servizi che applicano la condivisione delle risorse tra le origini (CORS). Per ulteriori informazioni, consulta Caricamento dell'SDK per JavaScript.

Puoi creare una build personalizzata dell'SDK in cui selezionare le versioni e i servizi Web specifici che desideri utilizzare. Quindi puoi scaricare il pacchetto SDK personalizzato per lo sviluppo locale ed eseguirne l'hosting per l'utilizzo da parte della tua applicazione. Per ulteriori informazioni sulla creazione di una build personalizzata dell'SDK, consulta Creazione dell'SDK per i browser.

Puoi scaricare le versioni distribuibili minimizzate e non minimizzate della versione corrente da: AWS SDK per JavaScript GitHub

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

Installazione mediante Bower

Bower è un gestore di pacchetti per il Web. Dopo aver installato Bower, puoi utilizzarlo per installare l'SDK. Per installare l'SDK mediante Bower, digita il seguente comando in una finestra del terminale:

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

Caricamento dell'SDK per JavaScript

La modalità di caricamento dell'SDK JavaScript dipende dal fatto che lo si stia caricando per eseguirlo in un browser Web o in Node.js.

Non tutti i servizi sono immediatamente disponibili nell'SDK. Per scoprire quali servizi sono attualmente supportati da AWS SDK per JavaScript, consulta https://github.com/aws/aws-sdk-js/.md blob/master/SERVICES

Node.js

Dopo aver installato l'SDK, puoi caricare il AWS pacchetto nell'applicazione node utilizzando. `require`

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

React Native

Per utilizzare l'SDK in un progetto React Native, installa prima l'SDK mediante npm:

```
npm install aws-sdk
```

Nella tua applicazione, fai riferimento alla versione dell'SDK compatibile con React Native utilizzando il seguente codice:

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

Browser

Il modo più rapido per iniziare a usare l'SDK è caricare il pacchetto SDK ospitato direttamente da Amazon Web Services. A questo scopo, aggiungi un elemento `<script>` alle pagine HTML in questa forma:

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

Per trovare l'attuale SDK_VERSION_NUMBER, consulta l'API Reference for the SDK disponibile nella API Reference Guide. JavaScript AWS SDK per JavaScript

Una volta caricato nella tua pagina, l'SDK è disponibile dalla variabile globale `AWS` (o `window.AWS`).

Se raggruppi il tuo codice e le dipendenze del modulo mediante browserify, puoi caricare l'SDK utilizzando `require`, in modo analogo a come si fa in Node.js.

Aggiornamento dell'SDK dalla versione 1 JavaScript

Le seguenti note ti aiutano ad aggiornare l'SDK JavaScript dalla versione 1 alla versione 2.

Conversione automatica dei tipi base64 e timestamp su input/output

L'SDK ora codifica e decodifica automaticamente i valori con codifica base64, nonché i valori timestamp, per conto dell'utente. Questa modifica interessa qualsiasi operazione in cui valori base64 o timestamp sono stati inviati tramite una richiesta o restituiti in una risposta che consente valori con codifica base64.

Il codice utente precedentemente convertito in base64 non è più necessario. I valori codificati come base64 vengono ora restituiti come buffer di oggetti da risposte del server e possono anche essere passati come buffer in entrata, Ad esempio, i seguenti parametri `SQS.sendMessage` della versione 1:

```
var params = {
```

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

Possono essere riscritti come segue.

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

Ecco come viene letto il messaggio.

```
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 spostato. RequestId a Response.requestId

L'SDK ora archivia la richiesta IDs per tutti i servizi in una posizione coerente sull'`response`oggetto, anziché all'interno della proprietà. `response.data` Ciò migliora la coerenza tra i servizi che espongono la richiesta IDs in modi diversi. Si tratta inoltre di una modifica di rilievo che rinomina la proprietà `response.data.RequestId` in `response.requestId` (`this.requestId` all'interno di una funzione di callback).

Nel tuo codice, modifica quanto segue:

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

```
});
```

alla seguente:

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

Elementi wrapper esposti

Se utilizzi `AWS.ElastiCache`, `AWS.RDS` o `AWS.Redshift`, per alcune operazioni devi accedere alla risposta tramite la proprietà di output di livello superiore della risposta.

Ad esempio, il metodo `RDS.describeEngineDefaultParameters` utilizzato per restituire quanto segue.

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

Ora restituisce il codice seguente.

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

L'elenco delle operazioni interessate per ciascun servizio è riportato nella seguente tabella.

Classe client	Operazioni

`AWS.ElastiCache`	`authorizeCacheSecurityGroupIngress`
	`createCacheCluster`
	`createCacheParameterGroup`
	`createCacheSecurityGroup`
	`createCacheSubnetGroup`
	`createReplicationGroup`
	`deleteCacheCluster`

Classe client	Operazioni
	deleteReplicationGroup describeEngineDefaultParameters modifyCacheCluster modifyCacheSubnetGroup modifyReplicationGroup purchaseReservedCacheNodesOffering rebootCacheCluster revokeCacheSecurityGroupIngress

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

Classe client	Operazioni

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

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

Classe client	Operazioni

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

Proprietà client interrotte

Le proprietà `.Client` e `.client` sono state rimosse dagli oggetti di servizio. Se utilizzi la proprietà `.Client` su una classe di servizio o una proprietà `.client` su un'istanza dell'oggetto di servizio, rimuovi queste proprietà dal tuo codice.

Il codice seguente utilizzato con la versione 1 dell'SDK per: JavaScript

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

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

Dovrebbe essere modificato come segue.

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

Configurazione dell'SDK per JavaScript

Prima di utilizzare l'SDK per JavaScript richiamare i servizi Web tramite l'API, è necessario configurare l'SDK. Come minimo, è necessario configurare queste impostazioni:

- La regione in cui sono richiesti i servizi.
- Le credenziali che autorizzano l'accesso alle risorse SDK.

Oltre a queste impostazioni, potresti dover configurare anche le autorizzazioni per le tue risorse. AWS Ad esempio, puoi limitare l'accesso a un bucket Amazon S3 o limitare una tabella Amazon DynamoDB per l'accesso in sola lettura.

La AWS SDKs and Tools Reference Guide contiene anche impostazioni, funzionalità e altri concetti fondamentali comuni a molti di essi. AWS SDKs

Gli argomenti di questa sezione descrivono vari modi per configurare l'SDK per Node.js e JavaScript per JavaScript eseguirlo in un browser Web.

Argomenti

Uso dell'oggetto configurazione globale

È possibile configurare l'SDK in due modi:

- Impostare l'oggetto configurazione globale tramite `AWS.Config`.

- Passare le informazioni di configurazione aggiuntive a un oggetto di servizio.

Impostare la configurazione globale con `AWS.Config` è spesso più semplice per iniziare, ma la configurazione a livello di servizio è in grado di garantire un maggiore controllo sui singoli servizi. La configurazione globale specificata da `AWS.Config` offre le impostazioni predefinite per gli oggetti di servizio creati successivamente, semplificando la loro configurazione. Tuttavia, è possibile aggiornare la configurazione di singoli oggetti di servizio quando le esigenze variano a seconda della configurazione globale.

Impostazione della Configurazione globale

Dopo aver caricato il pacchetto `aws-sdk` nel codice, puoi utilizzare la variabile globale `AWS` per accedere alle classi dell'SDK e interagire con i singoli servizi. L'SDK include un oggetto di configurazione globale, `AWS.Config`, che puoi utilizzare per specificare le impostazioni di configurazione dell'SDK necessarie per l'applicazione.

Configura l'SDK impostando le proprietà `AWS.Config` in base alle esigenze dell'applicazione. La tabella riportata di seguito riepiloga le proprietà `AWS.Config` comuni per impostare la configurazione dell'SDK.

Opzioni di configurazione	Description
`credentials`	Campo obbligatorio. Specifica le credenzia li usate per determinare l'accesso a servizi e risorse.
`region`	Campo obbligatorio. Specifica la regione in cui le richieste di servizi sono effettuate.
`maxRetries`	Opzionale. Specifica il numero massimo di volte in cui una determinata richiesta viene rieseguita.
`logger`	Opzionale. Specifica un oggetto logger su cui vengono scritte le informazioni di debug.
`update`	Opzionale. Aggiorna la configurazione corrente con i nuovi valori.

Per ulteriori informazioni sull'oggetto di configurazione, consulta `Class: AWS.Config`l'API Reference.

Esempi di configurazione globali

È necessario impostare la regione e le credenziali in `AWS.Config`. È possibile impostare queste proprietà come parte del costruttore `AWS.Config`, come illustrato nel seguente script di browser di esempio:

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

È anche possibile impostare queste proprietà dopo aver creato `AWS.Config` utilizzando il metodo `update`, come nell'esempio seguente, che aggiorna la regione:

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

Puoi ottenere le credenziali predefinite chiamando il metodo statico `getCredentials` di `AWS.config`:

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

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

Allo stesso modo, se avete impostato correttamente la regione nel `config` file, ottenete quel valore impostando che la variabile di `AWS_SDK_LOAD_CONFIG` ambiente sia impostata su un valore qualsiasi e chiamando la `region` proprietà statica di`AWS.config`:

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

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

Impostazione della configurazione per servizio

Ogni servizio utilizzato nell'SDK per JavaScript è accessibile tramite un oggetto di servizio che fa parte dell'API per quel servizio. Ad esempio, per accedere al servizio Amazon S3 è necessario creare l'oggetto di servizio Amazon S3. È possibile specificare le impostazioni di configurazione specifiche per un servizio come parte del costruttore per quell'oggetto di servizio. Quando si impostano i valori di configurazione su un oggetto di servizio, il costruttore acquisisce tutti i valori di configurazione utilizzati da `AWS.Config`, tra cui le credenziali.

Ad esempio, se devi accedere agli oggetti Amazon EC2 in più regioni, crea un oggetto di servizio Amazon EC2 per ogni regione e imposta di conseguenza la configurazione della regione di ciascun oggetto di servizio.

```
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'});
```

È anche possibile impostare valori di configurazione specifici per un servizio quando si configura l'SDK con `AWS.Config`. L'oggetto di configurazione globale supporta molte opzioni di configurazione specifica del servizio. Per ulteriori informazioni sulla configurazione specifica del servizio, consulta `Class: AWS.Config` l'API Reference. AWS SDK per JavaScript

Dati della configurazione immutabili

Le modifiche della configurazione globale si applicano alle richieste per tutti gli oggetti di servizio appena creati. Gli oggetti di servizio appena creati sono configurati con i dati dell'attuale configurazione globale prima e poi con le opzioni di configurazione locale. Gli aggiornamenti effettuati all'oggetto globale `AWS.config` non vengono applicati agli oggetti di servizio creati in precedenza.

Gli oggetti di servizio esistenti devono essere aggiornati manualmente con i nuovi dati di configurazione oppure è necessario creare e utilizzare un nuovo oggetto di servizio che dispone dei nuovi dati di configurazione. L'esempio seguente crea un nuovo oggetto di servizio Amazon S3 con nuovi dati di configurazione:

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

Impostazione della regione AWS

Una regione è un insieme denominato di AWS risorse nella stessa area geografica. Un esempio di regione è `us-east-1` la regione degli Stati Uniti orientali (Virginia settentrionale). Si specifica una regione durante la configurazione dell'SDK in JavaScript modo che l'SDK acceda alle risorse in quella regione. Alcuni servizi sono disponibili solo in Regioni specifiche.

L'SDK per JavaScript non seleziona una regione per impostazione predefinita. Tuttavia, è possibile impostare la regione utilizzando una variabile di ambiente, un file `config` condiviso o l'oggetto di configurazione globale.

In un costruttore della classe client

Quando si crea un'istanza di un oggetto di servizio, è possibile specificare la regione per la risorsa come parte del costruttore della classe client, come illustrato qui.

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

Uso dell'oggetto configurazione globale

Per impostare la regione nel JavaScript codice, aggiorna l'oggetto di configurazione `AWS.Config` globale come mostrato qui.

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

Per ulteriori informazioni sulle regioni attuali e sui servizi disponibili in ciascuna regione, consulta AWS Regioni ed endpoint in. Riferimenti generali di AWS

Utilizzo di una variabile di ambiente

È possibile impostare la regione utilizzando la variabile di ambiente `AWS_REGION`. Se definisci questa variabile, l'SDK for la JavaScript legge e la utilizza.

Utilizzo di un file di configurazione condiviso

Proprio come il file delle credenziali condivise consente di archiviare le credenziali per l'utilizzo da parte dell'SDK, è possibile conservare la regione e altre impostazioni di configurazione in un file condiviso denominato `config` utilizzato da. SDKs Se la variabile di `AWS_SDK_LOAD_CONFIG`

ambiente è stata impostata su un valore qualsiasi, l'SDK cerca JavaScript automaticamente un `config` file al momento del caricamento. Il percorso di salvataggio del file `config` varia a seconda del sistema operativo:

- Utenti Linux, macOS o Unix: `~/.aws/config`
- Utenti Windows: `C:\Users\USER_NAME\.aws\config`

Se non si dispone già di un file condiviso `config`, è possibile crearne uno nella directory designata. In questo esempio il file `config` imposta sia la regione sia il formato di output.

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

Per ulteriori informazioni sull'utilizzo dei file di configurazione e credenziali condivisi, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso la sezione File di configurazione e credenziali nella Guida per l'utente.AWS Command Line Interface

Ordine di precedenza per l'impostazione della regione

L'ordine di precedenza per l'impostazione della regione è il seguente:

- Se una regione è passata a un costruttore della classe client, viene utilizzata quella regione. In caso contrario, allora...
- Se una regione è impostata sull'oggetto di configurazione globale, viene utilizzata quella regione. In caso contrario, allora...
- Se la variabile di ambiente `AWS_REGION` è un valore truthy, viene utilizzata quella regione. In caso contrario, allora...
- Se la variabile di ambiente `AMAZON_REGION` è un valore truthy, viene utilizzata quella regione. In caso contrario, allora...
- Se la variabile di `AWS_SDK_LOAD_CONFIG` ambiente è impostata su un valore qualsiasi e il file delle credenziali condivise (`~/.aws/credentialso` il percorso indicato da`AWS_SHARED_CREDENTIALS_FILE`) contiene una regione per il profilo configurato, viene utilizzata tale regione. In caso contrario, allora...
- Se la variabile di `AWS_SDK_LOAD_CONFIG` ambiente è impostata su un valore qualsiasi e il file di configurazione (`~/.aws/configo` il percorso indicato da`AWS_CONFIG_FILE`) contiene una regione per il profilo configurato, viene utilizzata tale regione.

Specificazione degli endpoint personalizzati

Le chiamate ai metodi API nell'SDK for JavaScript vengono effettuate all'endpoint del servizio. URIs Per impostazione predefinita, questi endpoint vengono creati dalla regione configurata per il codice. Tuttavia, vi sono situazioni in cui è necessario specificare un endpoint personalizzato per le chiamate API.

Formato della stringa dell'endpoint

I valori dell'endpoint devono essere una stringa nel formato:

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

Endpoint per la regione ap-northeast-3

La `ap-northeast-3` regione in Giappone non viene restituita dall' APIsenumerazione Region, ad esempio. `EC2.describeRegions` Per definire gli endpoint per questa regione, segui il formato descritto in precedenza. Quindi l'endpoint Amazon EC2 per questa regione sarebbe

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

Endpoint per MediaConvert

È necessario creare un endpoint personalizzato con cui utilizzare. MediaConvert A ogni account cliente viene assegnato un endpoint, che è necessario utilizzare. Ecco un esempio di come utilizzare un endpoint personalizzato con. 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
};
```

Per ottenere l'endpoint API dell'account, consulta `MediaConvert.describeEndpoints` nella documentazione di riferimento delle API.

Assicurati di specificare la stessa regione nel codice della regione dell'URI dell'endpoint personalizzato. Una discrepanza tra l'impostazione della regione e l'URI dell'endpoint personalizzato può portare al fallimento delle chiamate API.

Per ulteriori informazioni MediaConvert, consulta la `AWS.MediaConvert`classe nell'API Reference o nella Guida per l'AWS Elemental MediaConvert utente.

Autenticazione SDK con AWS

È necessario stabilire in che modo il codice si autentica AWS durante lo sviluppo con. Servizi AWSÈ possibile configurare l'accesso programmatico alle AWS risorse in diversi modi a seconda dell'ambiente e dell' AWS accesso a disposizione.

Per scegliere il metodo di autenticazione e configurarlo per l'SDK, consulta Autenticazione e accesso nella Guida di riferimento agli strumenti AWS SDKs e agli strumenti.

Consigliamo ai nuovi utenti che si stanno sviluppando localmente e che non dispongono di un metodo di autenticazione da parte del datore di lavoro di AWS IAM Identity Center configurarlo. Questo metodo include l'installazione di AWS CLI per facilitare la configurazione e per accedere regolarmente al portale di AWS accesso. Se scegli questo metodo, l'ambiente dovrebbe contenere i seguenti elementi dopo aver completato la procedura per l'autenticazione di IAM Identity Center nella AWS SDKs and Tools Reference Guide:

- Il AWS CLI, che viene utilizzato per avviare una sessione del portale di AWS accesso prima di eseguire l'applicazione.
- Un AWS`config`file condiviso con un `[default]` profilo con un set di valori di configurazione a cui è possibile fare riferimento dall'SDK. Per trovare la posizione di questo file, consulta Posizione dei file condivisi nella Guida di riferimento agli strumenti AWS SDKs e strumenti.
- Il `config` file condiviso imposta l'`region`impostazione. Questo imposta l'impostazione predefinita Regione AWS utilizzata dall'SDK per AWS le richieste. Questa regione viene utilizzata per le richieste di servizio SDK che non sono specificate con una regione da utilizzare.
- L'SDK utilizza la configurazione del provider di token SSO del profilo per acquisire le credenziali prima di inviare richieste a. AWS Il `sso_role_name` valore, che è un ruolo IAM connesso a un set di autorizzazioni IAM Identity Center, consente l'accesso ai dati Servizi AWS utilizzati nell'applicazione.

  Il seguente `config` file di esempio mostra un profilo predefinito impostato con la configurazione del provider di token SSO. L'`sso_session`impostazione del profilo si riferisce alla `sso-`

sessionsezione denominata. La `sso-session` sezione contiene le impostazioni per avviare una sessione del portale di AWS accesso.

```
[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
```

L'SDK for JavaScript non necessita di pacchetti aggiuntivi (come `SSO` e`SSOOIDC`) da aggiungere all'applicazione per utilizzare l'autenticazione IAM Identity Center.

Avvia una sessione del portale di AWS accesso

Prima di eseguire un'applicazione che consente l'accesso Servizi AWS, è necessaria una sessione attiva del portale di AWS accesso affinché l'SDK utilizzi l'autenticazione IAM Identity Center per risolvere le credenziali. A seconda della durata della sessione configurata, l'accesso alla fine scadrà e l'SDK riscontrerà un errore di autenticazione. Per accedere al portale di AWS accesso, esegui il seguente comando in. AWS CLI

```
aws sso login
```

Se hai seguito le istruzioni e disponi di una configurazione predefinita del profilo, non è necessario richiamare il comando con un'`--profile`opzione. Se la configurazione del provider di token SSO utilizza un profilo denominato, il comando è `aws sso login --profile named-profile`.

Per verificare facoltativamente se hai già una sessione attiva, esegui il AWS CLI comando seguente.

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

Se la sessione è attiva, la risposta a questo comando riporta l'account IAM Identity Center e il set di autorizzazioni configurati nel `config` file condiviso.

> **Note**
>
> Se hai già una sessione attiva del portale di AWS accesso ed eseguiaws sso login, non ti verrà richiesto di fornire credenziali.
> La procedura di accesso potrebbe richiedere all'utente di consentire l' AWS CLI accesso ai dati. Poiché AWS CLI è basato sull'SDK per Python, i messaggi di autorizzazione potrebbero contenere variazioni del `botocore` nome.

Ulteriori informazioni di autenticazione

Utenti umani, noti anche come identità umane, sono le persone, gli amministratori, gli sviluppatori, gli operatori e i consumatori delle tue applicazioni. Devono avere un'identità per accedere agli AWS ambienti e alle applicazioni dell'utente. Gli utenti umani che fanno parte della tua organizzazione, ovvero tu, lo sviluppatore, sono noti come identità della forza lavoro.

Utilizza credenziali temporanee per l'accesso. AWS Puoi utilizzare un provider di identità per i tuoi utenti umani per fornire l'accesso federato agli AWS account assumendo ruoli che forniscono credenziali temporanee. Per la gestione centralizzata degli accessi, ti consigliamo di utilizzare AWS IAM Identity Center (IAM Identity Center) per gestire l'accesso ai tuoi account e le autorizzazioni all'interno di tali account. Per altre alternative, consulta quanto segue:

- Per ulteriori informazioni sulle best practice, consulta Best practice per la sicurezza in IAM nella Guida per l'utente di IAM.
- Per creare AWS credenziali a breve termine, consulta Temporary Security Credentials nella IAM User Guide.
- Per ulteriori informazioni su altri SDK per fornitori di JavaScript credenziali, consulta Fornitori di credenziali standardizzati nella and Tools Reference Guide.AWS SDKs

Impostazione delle credenziali

AWS utilizza le credenziali per identificare chi sta chiamando i servizi e se è consentito l'accesso alle risorse richieste.

Sia che venga eseguito in un browser Web o in un server Node.js, il JavaScript codice deve ottenere credenziali valide prima di poter accedere ai servizi tramite l'API. Le credenziali possono essere

configurate a livello globale sull'oggetto di configurazione utilizzando `AWS.Config`, oppure per servizio, passando le credenziali direttamente a un oggetto di servizio.

Esistono diversi modi per impostare le credenziali che differiscono tra Node.js e JavaScript nei browser Web. Gli argomenti in questa sezione descrivono come impostare le credenziali in Node.js o nei browser Web. In ogni caso, le opzioni sono riportate in ordine consigliato.

Best practice per le credenziali

L'impostazione corretta delle credenziali garantisce che l'applicazione o lo script di browser possano accedere ai servizi e alle risorse necessari, riducendo al minimo l'esposizione a problemi di sicurezza che possono inficiare le applicazioni mission critical o compromettere i dati sensibili.

Un importante principio da applicare durante l'impostazione delle credenziali è concedere sempre i privilegi minimi necessari per l'attività. È più sicuro fornire le autorizzazioni minime per le risorse e aggiungere ulteriori autorizzazioni in base alle esigenze, invece di fornire autorizzazioni che superano il privilegio minimo e, di conseguenza, dover risolvere i problemi di sicurezza scoperti più tardi. Ad esempio, a meno che non sia necessario leggere e scrivere singole risorse, come oggetti in un bucket Amazon S3 o una tabella DynamoDB, imposta tali autorizzazioni in modalità di sola lettura.

Per ulteriori informazioni sulla concessione del privilegio minimo, consulta la sezione Grant Least Privilege dell'argomento Best Practices nella IAM User Guide.

> **Warning**
>
> Mentre è possibile farlo, consigliamo di non effettuare l'hard coding delle credenziali all'interno di un'applicazione o di uno script di browser. La codifica rigida delle credenziali comporta il rischio di divulgare informazioni sensibili.

Per ulteriori informazioni su come gestire le chiavi di accesso, consulta Best practice per la gestione delle chiavi di AWS accesso nel. Riferimenti generali di AWS

Argomenti

- Impostazione delle credenziali su Node.js
- Impostazione delle credenziali in un browser Web

Impostazione delle credenziali su Node.js

Vi sono diversi modi su Node.js per fornire le credenziali all'SDK. Alcuni di questi sono più sicuri e altri offrono più comodità durante lo sviluppo di un'applicazione. Quando si ottengono le credenziali su Node.js, fai attenzione a ricorrere a più di una sorgente, ad esempio una variabile di ambiente e un file JSON caricati. È possibile modificare le autorizzazioni sotto cui il codice viene eseguito senza realizzare che la modifica è avvenuta.

Di seguito sono descritti i modi in cui è possibile fornire le proprie credenziali in ordine di raccomandazione:

1. Caricato da ruoli AWS Identity and Access Management (IAM) per Amazon EC2
2. Caricato dal file delle credenziali condiviso (`~/.aws/credentials`)
3. Caricato da variabili di ambiente
4. Caricato da un file JSON su disco
5. Altre classi di fornitori di credenziali fornite dall'SDK JavaScript

Se l'SDK dispone di più fonti di credenziali, la precedenza di selezione predefinita è la seguente:

1. Credenziali impostate esplicitamente attraverso il costruttore servizio-client
2. Variabili di ambiente
3. File delle credenziali condiviso
4. Credenziali caricate dal provider di credenziali ECS (se applicabile)
5. Credenziali ottenute utilizzando un processo di credenziali specificato nel file di AWS configurazione condiviso o nel file di credenziali condivise. Per ulteriori informazioni, consulta the section called “Credenziali che usano un processo di credenziali configurato”.
6. Credenziali caricate da AWS IAM utilizzando il provider di credenziali dell' EC2 istanza Amazon (se configurate nei metadati dell'istanza)

Per ulteriori informazioni, consulta `Class: AWS.Credentials`e `Class: AWS.CredentialProviderChain`nel riferimento all'API.

> **Warning**
>
> Sebbene sia possibile farlo, sconsigliamo di codificare le AWS credenziali nell'applicazione. Effettuare l'hard coding delle credenziali pone un rischio di esposizione dell'ID chiave di accesso e della chiave di accesso segreta.

Gli argomenti in questa sezione descrivono come caricare le credenziali su Node.js.

Argomenti

Caricamento delle credenziali in Node.js dai ruoli IAM per Amazon EC2

Se esegui l'applicazione Node.js su un' EC2 istanza Amazon, puoi sfruttare i ruoli IAM per Amazon per EC2 fornire automaticamente le credenziali all'istanza. Se configuri l'istanza per utilizzare i ruoli IAM, l'SDK seleziona automaticamente le credenziali IAM per l'applicazione, eliminando la necessità di fornire manualmente le credenziali.

Per ulteriori informazioni sull'aggiunta di ruoli IAM a un' EC2 istanza Amazon, consulta Using IAM roles for Amazon EC2 instances nella AWS SDKs and Tools Reference Guide.

Caricamento delle credenziali per una funzione Lambda Node.js

Quando si crea una AWS Lambda funzione, è necessario creare un ruolo IAM speciale con il permesso di eseguire la funzione. Questo ruolo si chiama ruolo di esecuzione. Quando configuri una funzione Lambda, devi specificare il ruolo IAM che hai creato come ruolo di esecuzione corrispondente.

Il ruolo di esecuzione fornisce alla funzione Lambda le credenziali necessarie per eseguire e richiamare altri servizi Web. Di conseguenza, non è necessario fornire credenziali per il codice Node.js scritto all'interno di una funzione Lambda.

Per ulteriori informazioni sulla creazione di un ruolo di esecuzione Lambda, consulta Manage Permissions: Using an IAM Role (Execution Role) nella Developer Guide.AWS Lambda

Caricamento delle credenziali su Node.js dal file delle credenziali condiviso

Puoi conservare i dati AWS delle tue credenziali in un file condiviso utilizzato da SDKs e nell'interfaccia a riga di comando. Quando l'SDK viene JavaScript caricato, cerca automaticamente nel file delle credenziali condivise, denominato «credenziali». Il percorso di salvataggio del file delle credenziali condiviso varia a seconda del sistema operativo:

- Su Linux, Unix e macOS il file delle credenziali condiviso è: `~/.aws/credentials`
- Su Windows il file delle credenziali condiviso è `C:\Users\USER_NAME\.aws\credentials`

Se non disponi già di un file delle credenziali condiviso, consulta Autenticazione SDK con AWS. Dopo aver seguito queste istruzioni, dovresti vedere un testo simile al seguente nel file delle credenziali, dove si *<YOUR_ACCESS_KEY_ID>* trova l'ID della tua chiave di accesso e la tua chiave di accesso *<YOUR_SECRET_ACCESS_KEY>* segreta:

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

Un esempio di utilizzo di questo file è disponibile in Nozioni di base su Node.js.

L'intestazione della sezione `[default]` specifica un profilo predefinito e i relativi valori per le credenziali. È possibile creare profili aggiuntivi nello stesso file di configurazione condiviso, ciascuno con le proprie informazioni sulle credenziali. L'esempio seguente mostra un file di configurazione con il profilo predefinito e due profili aggiuntivi:

```
[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>
```

Per impostazione predefinita, l'SDK verifica la variabile di ambiente `AWS_PROFILE` per determinare quale profilo utilizzare. Se la variabile `AWS_PROFILE` non è impostata nell'ambiente, l'SDK utilizza le credenziali per il profilo `[default]`. Per usare uno dei profili alternativi, modificare il valore della variabile di ambiente `AWS_PROFILE`. Ad esempio, dato il file di configurazione mostrato sopra, per utilizzare le credenziali dell'account di lavoro impostare la variabile di ambiente `AWS_PROFILE` su `work-account` (in funzione del sistema operativo in uso).

> **Note**
>
> Nell'impostare le variabili di ambiente, assicurarsi di procedere nel modo corretto (a seconda delle esigenze del sistema operativo) per rendere tali variabili disponibili nell'ambiente di shell o di comando.

Dopo aver impostato la variabile di ambiente (se necessario), puoi eseguire un JavaScript file che utilizza l'SDK, ad esempio un file denominato. `script.js`

```
$ node script.js
```

È anche possibile selezionare esplicitamente il profilo utilizzato dall'SDK, impostando `process.env.AWS_PROFILE` prima di caricare l'SDK oppure selezionando il provider di credenziali come nell'esempio seguente:

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

Caricamento delle credenziali su Node.js dalle variabili di ambiente

L'SDK rileva automaticamente AWS le credenziali impostate come variabili nell'ambiente e le utilizza per le richieste SDK, eliminando la necessità di gestire le credenziali nell'applicazione. Le variabili di ambiente impostate per fornire le credenziali sono:

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

Per maggiori dettagli sull'impostazione delle variabili di ambiente, consulta Supporto per le variabili di ambiente nella and Tools Reference Guide.AWS SDKs

Caricamento delle credenziali su Node.js da un file JSON

È possibile caricare la configurazione e le credenziali da un documento in formato JSON su disco utilizzando `AWS.config.loadFromPath`. Il percorso specificato è relativo alla directory di lavoro del processo. Ad esempio, per caricare le credenziali da un file `'config.json'` con il seguente contenuto:

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

Quindi usa il codice seguente:

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

> **Note**
>
> Il caricamento dei dati di configurazione da un documento JSON reimposta tutti i dati di configurazione esistenti. Aggiungi i dati di configurazione aggiuntivi dopo l'utilizzo di questa tecnica. Il caricamento delle credenziali da un documento in formato JSON non è supportato nello script di browser.

Caricamento delle credenziali su Node.js utilizzando un processo di credenziali configurato

Puoi originare credenziali utilizzando un metodo che non è integrato nell'SDK. A tale scopo, specificate un processo di creazione delle credenziali nel file di AWS configurazione condiviso o nel file delle credenziali condivise. Se la variabile di `AWS_SDK_LOAD_CONFIG` ambiente è impostata su un valore qualsiasi, l'SDK preferirà il processo specificato nel file di configurazione rispetto al processo specificato nel file delle credenziali (se presente).

Per i dettagli sulla specificazione di un processo di credenziali nel file di AWS configurazione condiviso o nel file delle credenziali condivise, consultate il AWS CLI Command Reference, in particolare le informazioni sull'approvvigionamento di credenziali da processi esterni.

Per ulteriori informazioni sull'utilizzo della variabile di ambiente `AWS_SDK_LOAD_CONFIG`, consulta the section called “Utilizzo di un file di configurazione condiviso” in questo documento.

Impostazione delle credenziali in un browser Web

Vi sono diversi modi per fornire le credenziali all'SDK dagli script di browser. Alcuni di questi sono più sicuri e altri offrono più comodità durante lo sviluppo di uno script. Di seguito sono descritti i modi in cui è possibile fornire le proprie credenziali in ordine di raccomandazione:

1. Utilizzo di Amazon Cognito Identity per autenticare gli utenti e fornire credenziali
2. Utilizzo delle identità della federazione delle identità Web
3. Effettuare l'hard coding nello script

> **Warning**
>
> Non è consigliabile codificare le AWS credenziali negli script. Effettuare l'hard coding delle credenziali pone un rischio di esposizione dell'ID chiave di accesso e della chiave di accesso segreta.

Argomenti

Utilizzo di Amazon Cognito Identity per autenticare gli utenti

Il modo consigliato per ottenere AWS le credenziali per gli script del browser consiste nell'utilizzare l'oggetto credenziali di Amazon Cognito Identity,. `AWS.CognitoIdentityCredentials` Amazon Cognito consente l'autenticazione degli utenti tramite provider di identità di terze parti.

Per utilizzare Amazon Cognito Identity, devi prima creare un pool di identità nella console Amazon Cognito. Un pool di identità rappresenta il gruppo di identità che l'applicazione fornisce agli utenti. Le identità fornite agli utenti identificano in modo univoco ogni account utente. Le identità di Amazon Cognito non sono credenziali. Vengono scambiate con credenziali utilizzando il supporto per la federazione delle identità web in (). AWS Security Token Service AWS STS

Amazon Cognito ti aiuta a gestire l'astrazione delle identità tra più provider di identità con l'oggetto. `AWS.CognitoIdentityCredentials` L'identità caricata viene quindi scambiata con le credenziali in AWS STS.

Configurazione dell'oggetto Amazon Cognito Identity Credentials

Se non ne hai ancora creato uno, crea un pool di identità da utilizzare con gli script del browser nella console Amazon Cognito prima della configurazione. `AWS.CognitoIdentityCredentials` Crea e associa ruoli IAM autenticati e non autenticati per il tuo pool di identità.

L'identità degli utenti non autenticati non è verificata. Ciò rende questo ruolo appropriato per utenti guest dell'app o per i casi in cui non importa se l'identità degli utenti è verificata. Gli utenti autenticati accedono all'applicazione tramite un provider di identità di terza parte che verifica la loro identità. Assicurati di creare l'ambito delle autorizzazioni di risorse in modo appropriato per evitare che utenti non autenticati possano accedere a esse.

Dopo aver configurato un pool di identità con i provider di identità associati, puoi utilizzare `AWS.CognitoIdentityCredentials` per autenticare gli utenti. Per configurare le credenziali dell'applicazione per utilizzare `AWS.CognitoIdentityCredentials`, imposta la proprietà `credentials` di `AWS.Config` o di una configurazione per servizio. Nell'esempio seguente viene utilizzato `AWS.Config`:

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

La proprietà `Logins` opzionale è una mappa di nomi di provider di identità ai token di identità per tali provider. Il modo in cui ottieni il token dal provider di identità dipende dal provider utilizzato. Ad esempio, se Facebook è uno dei provider di identità, puoi utilizzare la funzione `FB.login` di SDK di Facebook per ottenere un token del provider di identità:

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

Cambio degli utenti non autenticati in utenti autenticati

Amazon Cognito supporta utenti autenticati e non autenticati. Gli utenti non autenticati ottengono l'accesso alle risorse anche se non sono connessi con un provider di identità. Tale livello di accesso è utile per visualizzare i contenuti agli utenti prima che effettuino l'accesso. Ogni utente non autenticato ha un'identità unica in Amazon Cognito anche se non è stato effettuato l'accesso e l'autenticazione individualmente.

Utente inizialmente non autenticato

Gli utenti in genere iniziano con il ruolo non autenticato, per cui è possibile impostare la proprietà delle credenziali dell'oggetto di configurazione senza una proprietà `Logins`. In questo caso, la configurazione predefinita potrebbe essere simile alla seguente:

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

Cambio a utente autenticato

Quando un utente non autenticato accede a un provider di identità e dispone di un token, puoi cambiare l'utente da non autenticato ad autenticato chiamando una funzione personalizzata che aggiorna l'oggetto credenziali e aggiunge il token `Logins`:

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

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

```
}
```

Inoltre puoi creare un oggetto `CognitoIdentityCredentials`. In caso contrario, è necessario reimpostare le proprietà delle credenziali degli oggetti di servizio esistenti creati. Gli oggetti di servizio leggono dalla configurazione globale solo sull'inizializzazione dell'oggetto.

Per ulteriori informazioni sull'`CognitoIdentityCredentials`oggetto, consulta `AWS.CognitoIdentityCredentials`l'API Reference. AWS SDK per JavaScript

Utilizzo delle identità della federazione delle identità sul Web per autenticare gli utenti

È possibile configurare direttamente i singoli provider di identità per accedere alle AWS risorse utilizzando la federazione delle identità Web. AWS attualmente supporta l'autenticazione degli utenti tramite la federazione delle identità Web tramite diversi provider di identità:

- Login with Amazon
- Accedi con Facebook
- Accedi con Google

È necessario prima registrare l'applicazione con i provider supportati dall'applicazione. Successivamente, crea un ruolo IAM e configura le relative autorizzazioni. Il ruolo IAM che crei viene quindi utilizzato per concedere le autorizzazioni che hai configurato per esso tramite il rispettivo provider di identità. Ad esempio, puoi impostare un ruolo che consenta agli utenti che hanno effettuato l'accesso tramite Facebook di avere accesso in lettura a uno specifico bucket Amazon S3 che controlli.

Dopo aver ottenuto sia un ruolo IAM con privilegi configurati sia un'applicazione registrata con i provider di identità scelti, puoi configurare l'SDK per ottenere le credenziali per il ruolo IAM utilizzando il codice di supporto, come segue:

```
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
});
```

Il valore del parametro `ProviderId` dipende dal provider di identità specificato. Il valore del parametro `WebIdentityToken` è il token di accesso recuperato da un login riuscito con il provider

di identità. Per ulteriori informazioni su come configurare e recuperare i token di accesso per ogni provider di identità, consulta la documentazione per il provider di identità.

Fase 1: Registrazione con il provider di identità

Per iniziare, registra un'applicazione con i provider di identità che scegli di supportare. Ti verrà richiesto di fornire informazioni che identificano l'applicazione e il suo autore. In questo modo il provider di identità saprà chi è che riceve le informazioni sull'utente. In ogni caso, il provider di identità emetterà un ID dell'applicazione che è possibile utilizzare per configurare i ruoli utente.

Fase 2: Creazione di un ruolo IAM per un provider di identità

Dopo aver ottenuto l'ID dell'applicazione da un provider di identità, accedi alla console IAM all'indirizzo https://console.aws.amazon.com/iam/per creare un nuovo ruolo IAM.

Per creare un ruolo IAM per un provider di identità

1. Andare sulla sezione Roles (Ruoli) della console e scegliere Create New Role (Crea nuovo ruolo).
2. Digitare un nome per il nuovo ruolo che consente di tenere traccia del suo utilizzo, ad esempio **`facebookIdentity`**, quindi selezionare Next Step (Fase successiva).
3. In Select Role Type (Seleziona tipo di ruolo), scegliere Role for Identity Provider Access (Ruolo per accesso del provider di identità).
4. Per Grant access to web identity providers (Concedi l'accesso ai provider di identità Web), scegliere Select (Seleziona).
5. Dall'elenco degli Identity Provider, scegli il provider di identità che desideri utilizzare per questo ruolo IAM.

6. Digitare l'ID dell'applicazione fornito dal provider di identità in Application ID (ID dell'applicazione) e quindi scegliere Next Step (Fase successiva).
7. Configurare le autorizzazioni per le risorse che si desidera esporre, consentendo l'accesso a specifiche operazioni su risorse specifiche. Per ulteriori informazioni sulle autorizzazioni IAM, consulta Panoramica delle autorizzazioni AWS IAM nella Guida per l'utente IAM. Verificare e,

se necessario, personalizzare la relazione di trust del ruolo e quindi scegliere Next Step (Fase successiva).

8. Collegare altre policy necessarie e quindi scegliere Next Step (Fase successiva). Per ulteriori informazioni sulle policy IAM, consulta Panoramica delle policy IAM nella Guida per l'utente di IAM.
9. Verificare il nuovo ruolo, quindi selezionare Create Role (Crea ruolo).

Puoi fornire altri vincoli al ruolo, ad esempio definirne l'ambito per un utente specifico. IDs Se il ruolo consente di concedere autorizzazioni di scrittura per le risorse, assicurati di definire correttamente l'ambito del ruolo per gli utenti con privilegi corretti, altrimenti qualsiasi utente con identità Amazon, Facebook o Google sarà in grado di modificare le risorse nell'applicazione.

Per ulteriori informazioni sull'utilizzo della federazione delle identità Web in IAM, consulta About Web Identity Federation nella IAM User Guide.

Fase 3: Ottenere che un provider acceda ai Token dopo il login

Imposta l'azione di login per la tua applicazione utilizzando l'SDK del provider di identità. Puoi scaricare e installare un JavaScript SDK dal provider di identità che abilita l'accesso degli utenti, utilizzando uno dei due OAuth o OpenID. Per informazioni su come scaricare e configurare il codice dell'SDK dell'applicazione, consulta la documentazione sull'SDK per il tuo provider di identità:

- Login with Amazon
- Accedi con Facebook
- Accedi con Google

Fase 4: Ottenere credenziali temporanee

Dopo che l'applicazione, i ruoli, le autorizzazioni e le risorse sono stati configurati, aggiungi il codice alla tua applicazione per ottenere le credenziali temporanee. Queste credenziali vengono fornite AWS Security Token Service tramite la federazione delle identità web. Gli utenti accedono al provider di identità, che restituisce un token di accesso. Configura l'`AWS.WebIdentityCredentials`oggetto utilizzando l'ARN per il ruolo IAM che hai creato per questo provider di identità:

```
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
});
```

Gli oggetti di servizio creati successivamente avranno le credenziali corrette. Gli oggetti creati prima di impostare la proprietà `AWS.config.credentials` non dispongono delle credenziali correnti.

È anche possibile creare `AWS.WebIdentityCredentials` prima di recuperare il token di accesso. Questa operazione consente di creare gli oggetti di servizio che dipendono dalle credenziali prima di caricare il token di accesso. Per eseguire questa operazione, è necessario creare l'oggetto delle credenziali senza il parametro `WebIdentityToken`:

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

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

Quindi imposta `WebIdentityToken` nel callback dall'SDK del provider di identità che contiene il token di accesso:

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

Esempio di identità della federazione delle identità Web

Di seguito sono riportati alcuni esempi di utilizzo dell'identità federata Web per ottenere credenziali nel browser. JavaScript Questi esempi devono essere eseguiti da uno schema di host http:// o https:// per garantire che il provider di identità sia in grado di reindirizzare sull'applicazione.

Esempio di Login with Amazon

Il codice seguente mostra come usare Login con Amazon come provider di identità.

```
<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>
```

Esempio di accesso tramite Facebook

Il codice seguente mostra come usare l'accesso tramite Facebook come provider di identità:

```
<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>
```

Esempio di accesso tramite Google+

Il codice seguente mostra come usare l'accesso tramite Google+ come provider di identità. Il token di accesso utilizzato per la federazione delle identità Web da Google è archiviato in `response.id_token` anziché in `access_token` come qualsiasi altro provider di identità.

```
<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>
```

Blocco delle versioni dell'API

AWS i servizi dispongono di numeri di versione delle API per tenere traccia della compatibilità delle API. Le versioni delle API nei AWS servizi sono identificate da una stringa di data `YYYY-mm-dd` formattata. Ad esempio, l'attuale versione dell'API per Amazon S3 è. `2006-03-01`

Consigliamo di bloccare la versione dell'API per un servizio se il codice in produzione si basa su di essa. In questo modo è possibile isolare le applicazioni dalle modifiche del servizio risultanti da aggiornamenti dell'SDK. Se non si specifica una versione dell'API durante la creazione degli oggetti di servizio, l'SDK utilizza l'ultima versione dell'API per impostazione predefinita. Questo potrebbe far sì che l'applicazione faccia riferimento a un'API aggiornata con modifiche che incidono negativamente sull'applicazione.

Per bloccare la versione dell'API che utilizzi per un servizio, passa il parametro `apiVersion` quando crei l'oggetto di servizio. In questo esempio, un nuovo oggetto di servizio `AWS.DynamoDB` è bloccato per la versione dell'API `2011-12-05`:

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

È possibile configurare globalmente un set di versioni dell'API di servizio specificando il parametro `apiVersions` in `AWS.Config`. Ad esempio, per impostare versioni specifiche di DynamoDB e Amazon EC2 insieme all'attuale APIs API Amazon Redshift, imposta come segue: `apiVersions`

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

Ottenere le versioni dell'API

Per ottenere la versione dell'API per un servizio, consulta la sezione Blocco della versione dell'API nella pagina di riferimento del servizio, ad esempio https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html per Amazon S3.

Considerazioni su Node.js

Sebbene il codice Node.js lo sia JavaScript, l' AWS SDK per JavaScript utilizzo di Node.js può differire dall'utilizzo dell'SDK negli script del browser. Alcuni metodi API funzionano su Node.js ma non negli script di browser, e viceversa. E il corretto utilizzo di alcuni di APIs essi dipende dalla familiarità con i comuni schemi di codifica Node.js, come l'importazione e l'utilizzo di altri moduli Node.js come il modulo. `File System (fs)`

Utilizzo dei moduli di Node.js integrati

Node.js fornisce una raccolta di moduli integrati che è possibile utilizzare senza installazione. Per utilizzare questi moduli, è necessario creare un oggetto con il metodo `require` per specificare il nome del modulo. Ad esempio, per includere il modulo HTTP integrato, utilizza il seguente metodo.

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

Richiama i metodi del modulo come se fossero metodi di quell'oggetto. Ad esempio, qui c'è il codice per leggere un file HTML.

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

Per un elenco completo di tutti i moduli integrati che Node.js fornisce, consulta la documentazione su Node.js v6.11.1 sul sito Web di Node.js.

Utilizzo dei pacchetti NPM

Oltre ai moduli integrati, è possibile includere e incorporare il codice di terze parti da npm, il programma di gestione del pacchetto di Node.js. Si tratta di un repository di pacchetti Node.js open source e di un'interfaccia a riga di comando per installare i pacchetti. Per ulteriori informazioni su npm e un elenco dei pacchetti attualmente disponibili, vedere. https://www.npmjs.com Puoi anche saperne di più sui pacchetti Node.js aggiuntivi che puoi usare qui. GitHub

Un esempio di pacchetto npm che puoi usare con AWS SDK per JavaScript is`browserify`. Per informazioni dettagliate, vedi Creazione dell'SDK come dipendenza con Browserify. Un altro esempio è `webpack`. Per informazioni dettagliate, vedi Raggruppamento di applicazioni mediante webpack.

Argomenti

- Configurazione di maxSockets su Node.js

- Riutilizzo delle connessioni con Keep-Alive in Node.js
- Configurazione dei proxy per Node.js
- Registrazione dei bundle di certificati su Node.js

Configurazione di maxSockets su Node.js

Su Node.js, è possibile impostare il numero massimo di connessioni per origine. Se `maxSockets` è impostato, il client HTTP di basso livello mette in coda le richieste e le assegna ai socket non appena diventano disponibili.

In questo modo, è possibile impostare un limite superiore per il numero di richieste simultanee per una determinata origine effettuate alla volta. Impostando un valore basso è possibile ridurre il numero di errori di timeout o throttling ricevuti. Tuttavia, potrebbe aumentare l'utilizzo della memoria, perché le richieste vengono accodate fino a quando un socket diventa disponibile.

L'esempio seguente mostra come impostare `maxSockets` per tutti gli oggetti di servizio creati. Questo esempio consente fino a 25 connessioni simultanee a ciascun endpoint del servizio.

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

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

La stessa operazione può essere eseguita per servizio.

```
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
   }
});
```

Quando si utilizza l'opzione predefinita `https`, l'SDK acquisisce il valore `maxSockets` da `globalAgent`. Se il valore `maxSockets` non è definito o è `Infinity`, l'SDK assume un valore `maxSockets` di 50.

Per ulteriori informazioni sulla configurazione di `maxSockets` su Node.js, consulta la documentazione online su Node.js.

Riutilizzo delle connessioni con Keep-Alive in Node.js

Per impostazione predefinita, l' HTTP/HTTPS agente Node.js predefinito crea una nuova connessione TCP per ogni nuova richiesta. Per evitare il costo di stabilire una nuova connessione, è possibile riutilizzare una connessione esistente.

Per le operazioni di breve durata, ad esempio le query di Dynamo DB, il sovraccarico di latenza dell'impostazione di una connessione TCP potrebbe essere maggiore dell'operazione stessa. Inoltre, poiché la crittografia a riposo di DynamoDB è integrata AWS con KMS, è possibile che si verifichino delle latenze dovute al database che devono ristabilire nuove AWS KMS voci della cache per ogni operazione.

Il modo più semplice per configurare l'SDK per il riutilizzo delle connessioni TCP JavaScript consiste nell'impostare la variabile di ambiente su. `AWS_NODEJS_CONNECTION_REUSE_ENABLED 1` Questa caratteristica è stata aggiunta nella versione 2.463.0.

In alternativa, è possibile impostare la proprietà `keepAlive` di un agente HTTP o HTTPS impostato su `true`, come illustrato nell'esempio seguente.

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

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

L'esempio seguente mostra come impostare solo un `keepAlive` client DynamoDB:

```
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
  }
});
```

Se è abilitato `keepAlive`, è anche possibile impostare il ritardo iniziale per i pacchetti TCP Keep-Alive con `keepAliveMsecs`, che per impostazione predefinita è 1000ms. Vedere la documentazione di Node.js per i dettagli.

Configurazione dei proxy per Node.js

Se non riesci a connetterti direttamente a Internet, l'SDK for JavaScript supporta l'uso di proxy HTTP o HTTPS tramite un agente HTTP di terze parti, come proxy-agent. Per installare proxy-agent, digita quanto segue nella riga di comando.

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

Se si decide di utilizzare un proxy differente, segui prima le istruzioni per l'installazione e la configurazione per quel proxy. Per usare questo o un altro proxy di terze parti nella tua applicazione, è necessario impostare la proprietà `httpOptions` di `AWS.Config` per specificare il proxy scelto. Questo esempio mostra `proxy-agent`.

```
var AWS = require("aws-sdk");
var ProxyAgent = require('proxy-agent').ProxyAgent;
```

```
AWS.config.update({
  httpOptions: { agent: new ProxyAgent('http://internal.proxy.com') }
});
```

Per ulteriori informazioni sulle altre librerie di proxy, consulta npm, il programma di gestione dei pacchetti di Node.js.

Registrazione dei bundle di certificati su Node.js

Gli archivi di fiducia predefiniti per Node.js includono i certificati necessari per accedere ai servizi. AWS In alcuni casi, può essere preferibile includere solo uno specifico set di certificati.

In questo esempio, un determinato certificato su disco viene utilizzato per creare un `https.Agent` che respinge le connessioni a meno che non venga fornito il certificato designato. Il nuovo `https.Agent` viene poi utilizzato per aggiornare la configurazione dell'SDK.

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

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

Considerazioni sullo script di browser

I seguenti argomenti descrivono considerazioni speciali per l'utilizzo degli script nei browser. AWS SDK per JavaScript

Argomenti

- Creazione dell'SDK per i browser
- Cross-Origin Resource Sharing (CORS)

Creazione dell'SDK per i browser

L'SDK per JavaScript viene fornito come JavaScript file con supporto incluso per un set predefinito di servizi. Questo file viene in genere caricato negli script di browser utilizzando un tag `<script>` che fa riferimento al pacchetto SDK ospitato. Tuttavia, potrebbe essere necessario il supporto per i servizi diversi dal set predefinito o comunque si potrebbe rendere necessario personalizzare l'SDK.

Se utilizzi l'SDK al di fuori di un ambiente che applica CORS nel tuo browser e desideri accedere a tutti i servizi forniti dall'SDK per JavaScript, puoi creare una copia personalizzata dell'SDK localmente clonando il repository ed eseguendo gli stessi strumenti di compilazione che creano la versione ospitata predefinita dell'SDK. Le sezioni seguenti descrivono la procedura per creare l'SDK con servizi e versioni dell'API aggiuntivi.

Argomenti

Utilizzo di SDK Builder per creare l'SDK per JavaScript

Il modo più semplice per creare la propria build di AWS SDK per JavaScript è utilizzare l'applicazione web SDK Builder su js. https://sdk.amazonaws.com/builder/ Utilizza il builder di SDK per specificare i servizi e le versioni dell'API da includere nella build.

Scegli Select all services (Seleziona tutti i servizi) o Select default services (Seleziona servizi predefiniti) come punto di partenza da cui è possibile aggiungere o rimuovere i servizi. Scegli (Sviluppo) Development per il codice più leggibile oppure Minified (Minimizzato) per creare una build minimizzata da distribuire. Dopo aver scelto i servizi e le versioni da includere, scegli Build (Crea) per creare e scaricare l'SDK personalizzato.

Utilizzo della CLI per creare l'SDK per JavaScript

Per creare l'SDK per l' JavaScript utilizzo di AWS CLI, devi prima clonare il repository Git che contiene il codice sorgente SDK. È necessario avere installato Git e Node.js sul computer.

Innanzitutto, clona il repository da GitHub e modifica la directory nella directory:

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

Dopo aver clonato il repository, scarica i moduli di dipendenza sia per l'SDK sia per lo strumento di compilazione:

```
npm install
```

Ora puoi creare una versione in pacchetto dell'SDK.

Creazione dalla riga di comando

Lo strumento di creazione si trova in `dist-tools/browser-builder.js`. Esegui questo script digitando:

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

Questo comando crea il file aws-sdk.js. Questo file non è compresso. Per impostazione predefinita questo pacchetto include solo il set standard di servizi.

Minimizzate l'output della build

Per ridurre la quantità di dati sulla rete, JavaScript i file possono essere compressi tramite un processo chiamato minificazione. La minimizzazione rimuove commenti, spazi inutili e altri caratteri che facilitano leggibilità umana ma che non influiscono sull'esecuzione del codice. Lo strumento di compilazione è in grado di produrre output non compressi o minimizzati. Per creare l'output della build minimizzato, imposta la variabile di ambiente `MINIFY`:

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

Creazione di servizi e versioni dell'API specifici

È possibile selezionare quali servizi creare nell'SDK. Per selezionare i servizi, specifica i nomi del servizio, delimitati da virgole, come parametri. Ad esempio, per creare solo Amazon S3 e Amazon EC2, usa il seguente comando:

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

È inoltre possibile selezionare specifiche versioni dell'API della build dei servizi aggiungendo il nome della versione dopo il nome del servizio. Ad esempio, per creare entrambe le versioni API di Amazon DynamoDB, usa il seguente comando:

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

Gli identificatori di servizio e le versioni delle API sono disponibili nei file di configurazione specifici del servizio in/. https://github.com/aws/ aws-sdk-js tree/master/apis

Creazione di tutti i servizi

È possibile creare tutti i servizi e le versioni dell'API includendo il parametro `all`:

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

Creazione di servizi specifici

Per personalizzare il set selezionato di servizi inclusi nella build, passa la variabile di ambiente `AWS_SERVICES` al comando Browserify che contiene l'elenco di servizi desiderato. L'esempio seguente crea i servizi Amazon EC2, Amazon S3 e DynamoDB.

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

Creazione dell'SDK come dipendenza con Browserify

Node.js dispone di un meccanismo basato sul modulo per includere il codice e la funzionalità da sviluppatori di terze parti. Questo approccio modulare non è supportato nativamente dall'esecuzione nei browser Web. JavaScript Tuttavia, con uno strumento chiamato Browserify, è possibile utilizzare l'approccio del modulo Node.js e i moduli scritti per Node.js nel browser. Browserify crea le dipendenze del modulo per uno script del browser in un unico JavaScript file autonomo che è possibile utilizzare nel browser.

È possibile creare l'SDK come dipendenza della libreria per qualsiasi script di browser utilizzando Browserify. Ad esempio, il codice Node.js riportato di seguito richiede l'SDK:

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

Questo codice di esempio può essere compilato in una versione compatibile con il browser utilizzando Browserify:

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

L'applicazione, incluse le dipendenze SDK, è disponibile all'interno del browser tramite `browser-app.js`.

Per ulteriori informazioni su Browserify, consulta il sito Web di Browserify.

Cross-Origin Resource Sharing (CORS)

Il Cross-origin resource sharing, o CORS, è una caratteristica di sicurezza dei moderni browser Web. In questo modo i browser Web possono negoziare quali domini possono fare richieste di siti Web o servizi esterni. CORS è fondamentale quando si sviluppano applicazioni di tipo browser con AWS SDK per JavaScript perché la maggior parte delle richieste di risorse vengono inviate a un dominio esterno, ad esempio l'endpoint di un servizio Web. Se JavaScript l'ambiente utilizza la sicurezza CORS, è necessario configurare CORS con il servizio.

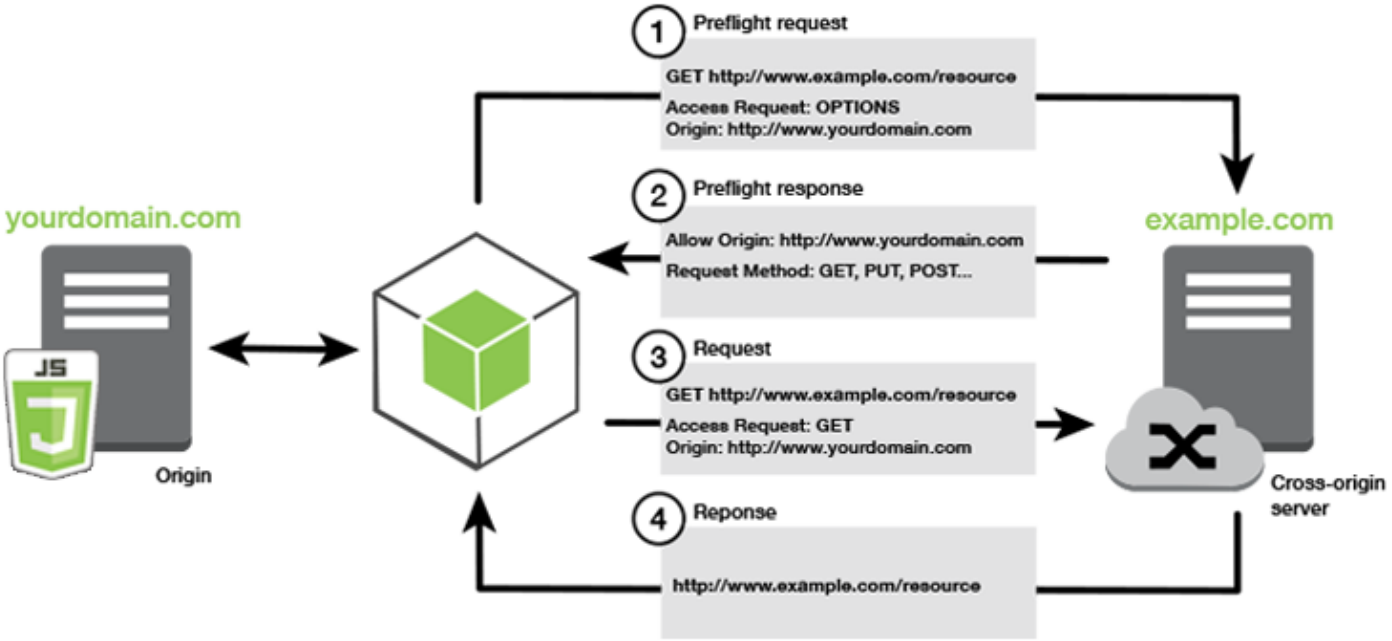
CORS determina se consentire la condivisione di risorse in una richiesta cross-origin in base a quanto segue:

- Il dominio specifico che effettua la richiesta
- Il tipo di richiesta HTTP effettuata (GET, PUT, POST, DELETE e così via)

Come funziona CORS

Nel caso più semplice, lo script di browser invia una richiesta GET per una risorsa da un server in un altro dominio. A seconda della configurazione CORS del server, se la richiesta proviene da un dominio che è autorizzato a inviare le richieste GET, il server cross-origin risponde restituendo la risorsa richiesta.

Se il dominio che effettua la richiesta o il tipo di richiesta HTTP non è autorizzato, la richiesta viene negata. Tuttavia, CORS permette di preparare la richiesta prima dell'invio. In questo caso, viene effettuata una richiesta preliminare in cui viene inviata la richiesta di accesso `OPTIONS`. Se la configurazione CORS del server cross-origin consente di concedere l'accesso al dominio richiedente, il server invia una risposta preliminare che elenca tutti i tipi di richieste HTTP che il dominio può effettuare sulla risorsa richiesta.



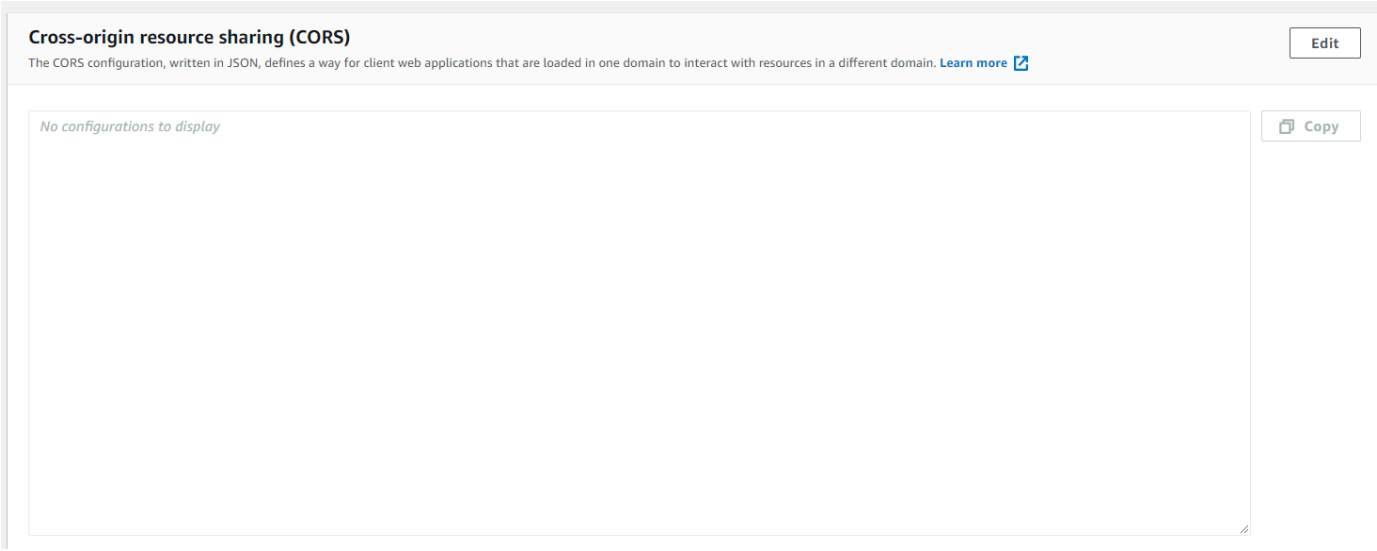
Si richiede la configurazione CORS

I bucket Amazon S3 richiedono la configurazione CORS prima di poter eseguire operazioni su di essi. In alcuni JavaScript ambienti CORS potrebbe non essere applicato e pertanto la configurazione di CORS non è necessaria. Ad esempio, se ospiti l'applicazione da un bucket Amazon S3 e accedi alle risorse da `*.s3.amazonaws.com` o da qualche altro endpoint specifico, le tue richieste non accederanno a un dominio esterno. Pertanto, questa configurazione non richiede CORS. In questo caso, CORS viene ancora utilizzato per servizi diversi da Amazon S3.

Configurazione di CORS per un bucket Amazon S3

Puoi configurare un bucket Amazon S3 per utilizzare CORS nella console Amazon S3.

1. Nella console Amazon S3, scegli il bucket che desideri modificare.
2. Seleziona la scheda Autorizzazioni e scorri verso il basso fino al pannello Cross-Origin Resource Sharing (CORS).



3. Scegli Modifica e digita la tua configurazione CORS nell'editor di configurazione CORS, quindi scegli Salva.

Una configurazione CORS è un file XML che contiene una serie di regole all'interno di un `<CORSRule>`. Una configurazione può avere massimo 100 regole. Una regola è definita da uno dei seguenti tag:

- `<AllowedOrigin>`, che consente di specificare le origini di dominio a cui si consente di effettuare richieste multidominio.
- `<AllowedMethod>`, che specifica un tipo di richiesta consentita (GET, POST, PUT, DELETE, HEAD) nelle richieste multidominio.
- `<AllowedHeader>`, che specifica le intestazioni consentite in una richiesta OPTIONS preliminare.

Per configurazioni di esempio, vedi Come posso configurare CORS su My Bucket? nella Guida per l'utente di Amazon Simple Storage Service.

Esempio di configurazione CORS

Il seguente esempio di configurazione CORS consente a un utente di visualizzare, aggiungere, eliminare o aggiornare gli oggetti all'interno di un bucket dal dominio `example.org`, anche se ti consigliamo di creare l'ambito `<AllowedOrigin>` del dominio del sito Web. È possibile specificare `"*"` per consentire l'origine.

> **Important**
>
> Nella nuova console S3, la configurazione CORS deve essere JSON.

XML

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

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "https://www.example.org"
        ],
        "ExposeHeaders": [
             "ETag",
             "x-amz-meta-custom-header"]
```

```
    }
]
```

Questa configurazione non autorizza l'utente a eseguire azioni nel bucket. Abilita il modello di sicurezza del browser per consentire una richiesta ad Amazon S3. Le autorizzazioni devono essere configurate tramite i permessi dei bucket o i permessi dei ruoli IAM.

Puoi utilizzarlo `ExposeHeader` per consentire all'SDK di leggere le intestazioni di risposta restituite da Amazon S3. Ad esempio, se si desidera leggere l'intestazione `ETag` da un `PUT` o un caricamento in più parti, è necessario includere il tag `ExposeHeader` nella configurazione, come mostrato nell'esempio precedente. Il kit SDK è in grado di accedere solo alle intestazioni esposte attraverso la configurazione CORS. Se si impostano i metadati nell'oggetto, i valori vengono restituiti come intestazioni con il prefisso `x-amz-meta-`, ad esempio `x-amz-meta-my-custom-header`, e devono essere esposti in modo analogo.

Raggruppamento di applicazioni mediante webpack

Le applicazioni Web negli script del browser o l'utilizzo da parte di Node.js dei moduli di codice crea dipendenze. Questi moduli di codice possono avere dipendenze proprie, generando una raccolta di moduli interconnessi richiesti dall'applicazione per funzionare. Per gestire le dipendenze, puoi utilizzare un module bundler come webpack.

Il module bundler webpack analizza il codice dell'applicazione, cercando le istruzioni `import` o `require`, per creare pacchetti che contengano tutti gli asset di cui l'applicazione ha bisogno affinché gli asset possano essere facilmente forniti attraverso una pagina Web. L'SDK per JavaScript può essere incluso nel webpack come una delle dipendenze da includere nel pacchetto di output.

Per ulteriori informazioni su webpack, consulta il bundler del modulo webpack su. GitHub

Installazione di webpack

Per installare il module bundler webpack, devi installare prima npm, il gestore di pacchetti Node.js. Digita il seguente comando per installare la CLI JavaScript e il modulo webpack.

```
npm install webpack
```

Potrebbe anche essere necessario installare un plugin di webpack che consente di caricare i file JSON. Digita il comando seguente per installare il plugin per il caricamento di file JSON.

```
npm install json-loader
```

Configurazione di webpack

Per impostazione predefinita, webpack cerca un JavaScript file denominato `webpack.config.js` nella directory principale del progetto. Questo file specifica le opzioni di configurazione. Ecco un esempio di file di configurazione `webpack.config.js`.

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

In questo esempio, è specificato `browser.js` come punto di ingresso. Il punto di ingresso è il file utilizzato da webpack per avviare la ricerca dei moduli importati. Come `bundle.js` è specificato il nome del file di output. Questo file di output conterrà tutto ciò di cui JavaScript l'applicazione ha bisogno per funzionare. Se il codice specificato nel punto di ingresso importa o richiede altri moduli,

come l'SDK for JavaScript, quel codice viene fornito in bundle senza bisogno di specificarlo nella configurazione.

La configurazione nel plugin `json-loader` che è stato installato in precedenza specifica a webpack come importare i file JSON. Per impostazione predefinita, webpack supporta solo i caricatori JavaScript ma utilizza i caricatori per aggiungere il supporto per l'importazione di altri tipi di file. Poiché l'SDK for JavaScript fa un uso estensivo dei file JSON, webpack genera un errore durante la generazione del pacchetto se non è incluso. `json-loader`

Esecuzione di webpack

Per creare un'applicazione per l'utilizzo di webpack, aggiungi il codice seguente all'oggetto `scripts` nel tuo file `package.json`.

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

Ecco un esempio di `package.json` che mostra l'aggiunta di webpack.

```
{
  "name": "aws-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "aws-sdk": "^2.6.1"
  },
  "devDependencies": {
    "json-loader": "^0.5.4",
    "webpack": "^1.13.2"
  }
}
```

Per creare l'applicazione, digita il comando riportato di seguito.

```
npm run build
```

Il bundler del modulo webpack genera quindi il JavaScript file specificato nella directory principale del progetto.

Utilizzo del bundle di webpack

Per utilizzare il bundle in uno script del browser, puoi incorporare il bundle utilizzando un tag `<script>` come mostrato nel seguente esempio.

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

Importazione di singoli servizi

Uno dei vantaggi di webpack è che analizza le dipendenze nel tuo codice e raggruppa solo il codice richiesto dalla tua applicazione. Se utilizzi l'SDK per JavaScript, il raggruppamento solo delle parti dell'SDK effettivamente utilizzate dall'applicazione può ridurre notevolmente le dimensioni dell'output del webpack.

Considera il seguente esempio del codice utilizzato per creare un oggetto di servizio Amazon S3.

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

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

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

La funzione `require()` specifica l'intero SDK. Un pacchetto webpack generato con questo codice includerebbe l'SDK completo, ma l'SDK completo non è richiesto quando viene utilizzata solo la classe client Amazon S3. La dimensione del pacchetto sarebbe notevolmente inferiore se fosse

inclusa solo la parte dell'SDK richiesta per il servizio Amazon S3. Anche l'impostazione della configurazione non richiede l'SDK completo perché puoi impostare i dati di configurazione sull'oggetto del servizio Amazon S3.

Ecco come appare lo stesso codice quando include solo la parte Amazon S3 dell'SDK.

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

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

Raggruppamento per Node.js

Puoi utilizzare webpack per generare bundle eseguiti in Node.js specificandolo come destinazione nella configurazione.

```
target: "node"
```

Questa funzione è utile quando si esegue un'applicazione Node.js in un ambiente in cui lo spazio su disco è limitato. Ecco un esempio di configurazione `webpack.config.js` con Node.js specificato come destinazione dell'output.

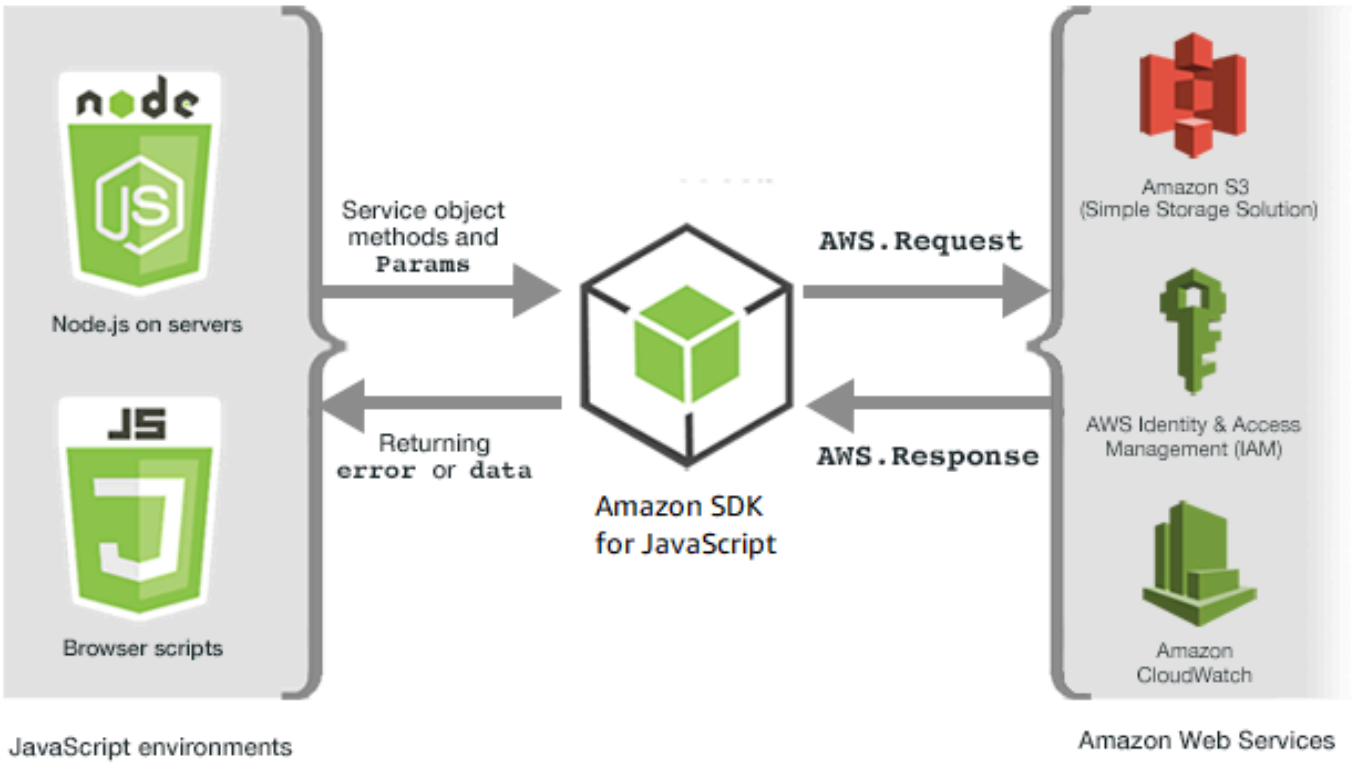
```
// Import path for resolving file paths
var path = require('path');
module.exports = {
  // Specify the entry point for our app
  entry: [
    path.join(__dirname, 'node.js')
  ],
  // Specify the output file containing our bundled code
  output: {
    path: __dirname,
    filename: 'bundle.js'
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

Utilizzo dei servizi nell'SDK per JavaScript

AWS SDK per JavaScript Fornisce l'accesso ai servizi che supporta attraverso una raccolta di classi client. Da queste classi client, si creano oggetti di interfaccia di servizio, comunemente chiamati oggetti di servizio. Ogni AWS servizio supportato include una o più classi client che offrono funzionalità e risorse APIs di servizio di basso livello. Ad esempio, Amazon APIs DynamoDB è disponibile tramite la classe. `AWS.DynamoDB`

I servizi esposti tramite l'SDK JavaScript seguono lo schema di richiesta-risposta per lo scambio di messaggi con le applicazioni chiamanti. In questo schema, il codice che richiama un servizio invia una HTTP/HTTPS richiesta a un endpoint per il servizio. La richiesta contiene i parametri necessari per richiamare correttamente la funzionalità specifica chiamata. Il servizio richiamato genera una risposta che viene inviata nuovamente al richiedente. La risposta contiene dati se l'operazione ha avuto esito positivo o informazioni di errore se l'operazione non ha avuto esito positivo.



L'invocazione AWS di un servizio include l'intero ciclo di vita della richiesta e della risposta di un'operazione su un oggetto del servizio, inclusi eventuali nuovi tentativi. Una richiesta viene incapsulata nell'SDK dall'oggetto `AWS.Request`. La risposta è incapsulata nell'SDK dall'`AWS.Response`oggetto, che viene fornito al richiedente tramite una delle diverse tecniche, ad esempio una funzione di callback o una promessa. JavaScript

Argomenti

Creazione e chiamata di oggetti di servizio

L' JavaScript API supporta la maggior parte dei AWS servizi disponibili. Ogni classe di servizio nell' JavaScript API fornisce l'accesso a tutte le chiamate API del relativo servizio. Per ulteriori informazioni sulle classi di servizio, le operazioni e i parametri dell' JavaScript API, consulta il riferimento all'API.

Quando utilizzi l'SDK in Node.js, puoi aggiungere il pacchetto SDK alla tua applicazione utilizzando `require`, che fornisce supporto per tutti i servizi correnti.

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

Quando si utilizza l'SDK con il browser JavaScript, si carica il pacchetto SDK negli script del browser utilizzando il pacchetto SDK ospitato da AWS. Per caricare il pacchetto SDK, aggiungi il seguente elemento `<script>`:

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

Per trovare l'attuale SDK_VERSION_NUMBER, consulta l'API Reference for the SDK consultabile nella API Reference Guide. JavaScript AWS SDK per JavaScript

Il pacchetto SDK ospitato di default fornisce il supporto per un sottoinsieme dei servizi disponibili. AWS Per un elenco dei servizi predefiniti nel pacchetto SDK in hosting per il browser, consulta la sezione relativa ai servizi supportati nella documentazione di riferimento delle API. Puoi utilizzare l'SDK con altri servizi se il controllo di sicurezza della funzionalità CORS è disabilitato. In questo caso, puoi creare una versione personalizzata dell'SDK per includere i servizi aggiuntivi necessari. Per ulteriori informazioni sulla creazione di una versione personalizzata dell'SDK, consulta Creazione dell'SDK per i browser.

Richiesta di singoli servizi

La richiesta dell'SDK per JavaScript come mostrato in precedenza include l'intero SDK nel codice. In alternativa, puoi scegliere di richiedere solo i singoli servizi utilizzati dal tuo codice. Considera il seguente codice utilizzato per creare un oggetto di servizio Amazon S3.

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

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

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

Nell'esempio precedente, la funzione `require` specifica l'intero SDK. La quantità di codice da trasportare sulla rete e il sovraccarico di memoria del codice sarebbero notevolmente inferiori se fosse inclusa solo la parte dell'SDK richiesta per il servizio Amazon S3. Per richiedere un singolo servizio, chiama la funzione `require` come illustrato, includendo il costruttore del servizio a lettere minuscole.

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

Ecco come appare il codice per creare il precedente oggetto di servizio Amazon S3 quando include solo la parte Amazon S3 dell'SDK.

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

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

Puoi comunque accedere al AWS namespace globale senza tutti i servizi ad esso collegati.

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

Si tratta di una tecnica utile quando si applica la stessa configurazione su più singoli servizi, ad esempio per fornire le stesse credenziali a tutti i servizi. La richiesta di singoli servizi dovrebbe ridurre il tempo di caricamento e il consumo di memoria in Node.js. Quando la richiesta dei singoli servizi viene eseguita utilizzando uno strumento di raggruppamento come Browserify o webpack, l'SDK sarà una frazione delle dimensioni complete. Ciò è utile in ambienti con limiti di memoria o spazio su disco come un dispositivo IoT o in una funzione Lambda.

Creazione di oggetti di servizio

Per accedere alle funzionalità del servizio tramite l' JavaScript API, è innanzitutto necessario creare un oggetto di servizio tramite il quale accedere a un set di funzionalità fornite dalla classe client sottostante. Generalmente per ciascun servizio viene fornita una classe client; tuttavia, alcuni servizi distribuiscono l'accesso alle loro funzionalità tra più classi client.

Per utilizzare una funzionalità, devi creare un'istanza della classe che fornisce l'accesso a tale funzionalità. L'esempio seguente mostra la creazione di un oggetto servizio per DynamoDB dalla `AWS.DynamoDB` classe client.

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

Per impostazione predefinita, un oggetto di servizio viene configurato con le impostazioni globali utilizzate anche per configurare l'SDK. Tuttavia, puoi configurare un oggetto di servizio con i dati di configurazione di runtime specifici per l'oggetto di servizio. I dati di configurazione specifici per il servizio vengono applicati dopo aver applicato le impostazioni di configurazione globali.

Nell'esempio seguente, un oggetto di servizio Amazon EC2 viene creato con la configurazione per una regione specifica, ma per il resto utilizza la configurazione globale.

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

Oltre a supportare la configurazione specifica del servizio applicata a un singolo oggetto di servizio, puoi anche applicare la configurazione specifica del servizio a tutti gli oggetti di servizio appena creati per una determinata classe. Ad esempio, per configurare tutti gli oggetti di servizio creati dalla classe Amazon EC2 per utilizzare la regione US West (Oregon) (`us-west-2`), aggiungi quanto segue all'oggetto di configurazione `AWS.config` globale.

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

Blocco della versione API di un oggetto di servizio

Puoi bloccare un oggetto di servizio su una determinata versione API di un servizio specificando l'opzione `apiVersion` durante la creazione dell'oggetto. Nell'esempio seguente, viene creato un oggetto servizio DynamoDB bloccato su una versione API specifica.

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

Per ulteriori informazioni sul blocco della versione API di un oggetto di servizio, consulta Blocco delle versioni dell'API.

Specifica dei parametri dell'oggetto di servizio

Quando chiami un metodo di un oggetto di servizio, trasferisci i parametri in JSON come richiesto dall'API. Ad esempio, in Amazon S3, per ottenere un oggetto per un bucket e una chiave specificati, passa i seguenti parametri al metodo. `getObject` Per ulteriori informazioni sul trasferimento di parametri JSON, consulta Utilizzo di JSON.

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

Per ulteriori informazioni sui parametri di Amazon S3, consulta il riferimento `Class: AWS.S3`all'API.

Inoltre, puoi associare valori ai singoli parametri durante la creazione di un oggetto di servizio utilizzando il parametro `params`. Il valore del parametro `params` degli oggetti di servizio è una mappa che specifica uno o più valori del parametro definiti dall'oggetto di servizio. L'esempio seguente mostra il `Bucket` parametro di un oggetto di servizio Amazon S3 associato a un bucket denominato. `amzn-s3-demo-bucket`

```
var s3bucket = new AWS.S3({params: {Bucket: 'amzn-s3-demo-bucket'}, apiVersion:
 '2006-03-01' });
```

Associando l'oggetto di servizio a un bucket, l'oggetto di servizio `s3bucket` tratta il valore del parametro `amzn-s3-demo-bucket` come un valore predefinito che non deve più essere specificato per le operazioni successive. Tutti i valori del parametro associati vengono ignorati quando si utilizza l'oggetto per operazioni in cui il valore del parametro non è applicabile. Puoi sostituire questo parametro associato quando effettui chiamate sull'oggetto di servizio specificando un nuovo valore.

```
var s3bucket = new AWS.S3({ params: {Bucket: 'amzn-s3-demo-bucket'}, apiVersion:
 '2006-03-01' });
```

```
s3bucket.getObject({Key: 'keyName'});
// ...
s3bucket.getObject({Bucket: 'amzn-s3-demo-bucket3', Key: 'keyOtherName'});
```

I dettagli sui parametri disponibili per ciascun metodo si trovano nella documentazione di riferimento delle API.

Registrazione delle chiamate AWS SDK per JavaScript

AWS SDK per JavaScript È dotato di un logger integrato in modo da poter registrare le chiamate API effettuate con l'SDK per. JavaScript

Per attivare il logger e stampare le voci di registro nella console, aggiungi la seguente istruzione al tuo codice.

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

Di seguito è riportato un esempio di output del log.

```
[AWS s3 200 0.185s 0 retries] createMultipartUpload({ Bucket: 'amzn-s3-demo-logging-
bucket', Key: 'issues_1704' })
```

Utilizzo di un logger di terze parti

Puoi anche utilizzare un logger di terze parti, a condizione che sia dotato di operazioni `write()` o `log()` per la scrittura in un file di log o su un server. È necessario installare e configurare il logger personalizzato come indicato prima di poterlo utilizzare con l'SDK for. JavaScript

Uno di questi logger che è possibile utilizzare in entrambi gli script del browser o in Node.js è logplease. In Node.js, puoi configurare logplease per scrivere le voci di log in un file di log. Puoi anche utilizzarlo con webpack.

Quando utilizzi un logger di terze parti, imposta tutte le opzioni prima di assegnarlo a `AWS.Config.logger`. Ad esempio, quanto segue specifica un file di log esterno e imposta il livello di log per logplease

```
// Require AWS Node.js SDK
const AWS = require('aws-sdk')
// Require logplease
const logplease = require('logplease');
```

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

Per ulteriori informazioni su logplease, consulta logplease Simple Logger su. JavaScript GitHub

Chiamate asincrone dei servizi

Tutte le richieste effettuate attraverso l'SDK sono asincrone. Questo è importante da tenere a mente quando si scrivono gli script del browser. JavaScript l'esecuzione in un browser Web in genere ha un solo thread di esecuzione. Dopo aver effettuato una chiamata asincrona a un AWS servizio, lo script del browser continua a essere eseguito e durante il processo può provare a eseguire il codice che dipende da quel risultato asincrono prima che venga restituito.

L'esecuzione di chiamate asincrone a un AWS servizio include la gestione di tali chiamate in modo che il codice non tenti di utilizzare i dati prima che i dati siano disponibili. Gli argomenti di questa sezione spiegano la necessità di gestire le chiamate asincrone e illustrano diverse tecniche che è possibile utilizzare per gestirle.

Argomenti

- Gestione delle chiamate asincrone
- Utilizzo di una funzione di callback anonimo
- Utilizzo di un listener di eventi oggetto di richiesta
- Utilizzo di async/await
- Usare Promises JavaScript

Gestione delle chiamate asincrone

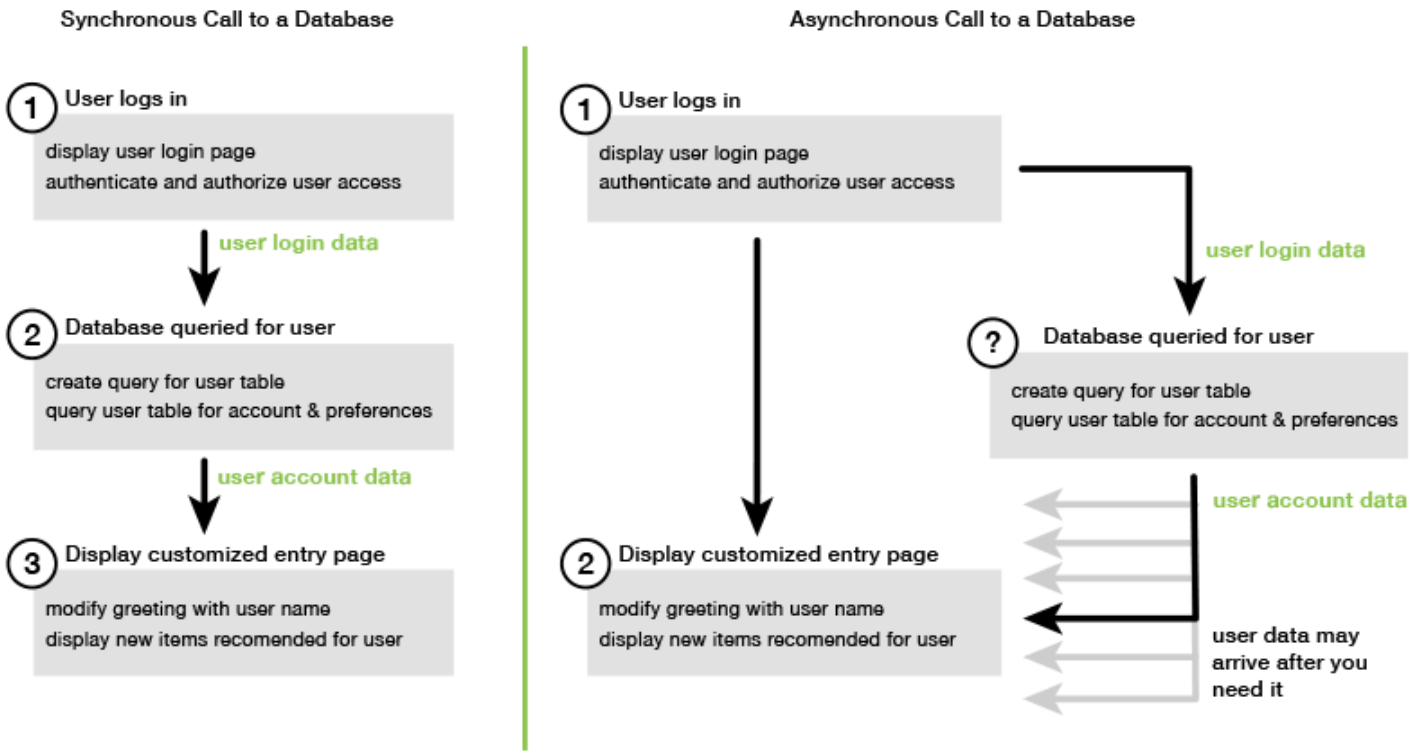
Ad esempio, la home page di un sito Web di e-commerce consente l'accesso ai clienti registrati. Parte del vantaggio per i clienti che effettuano l'accesso è che, dopo l'accesso, il sito si adatta alle loro preferenze specifiche. Per fare in modo che ciò accada:

1. Il cliente deve accedere ed essere convalidato con le proprie credenziali di accesso.

2. Le preferenze del cliente vengono richieste da un database dei clienti.
3. Il database fornisce le preferenze del cliente che vengono utilizzate per personalizzare il sito prima che la pagina venga caricata.

Se queste attività vengono eseguite in modo sincrono, ognuna deve terminare prima venga avviata quella successiva. La pagina Web non è in grado di completare il caricamento finché le preferenze del cliente non vengono restituite dal database. Tuttavia, dopo che la query del database viene inviata al server, la ricezione dei dati del cliente può essere posticipata o può addirittura fallire a causa di colli di bottiglia della rete, traffico di database eccezionalmente elevato o connessione scadente dei dispositivi mobili.

Per impedire il congelamento del sito Web in tali condizioni, chiama il database in modo asincrono. Dopo l'esecuzione della chiamata al database, inviando la tua richiesta asincrona, il tuo codice continua a essere eseguito come previsto. Se non gestisci correttamente la risposta di una chiamata asincrona, il tuo codice può tentare di utilizzare le informazioni che si aspetta dal database quando tali dati non sono ancora disponibili.



Utilizzo di una funzione di callback anonimo

Ogni metodo dell'oggetto di servizio che crea un oggetto `AWS.Request` può accettare una funzione di callback anonimo come ultimo parametro. La firma di questa funzione di callback è:

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

Questa funzione di callback viene eseguita quando vengono restituiti una risposta corretta o i dati dell'errore. Se la chiamata al metodo va a buon fine, i contenuti della risposta sono disponibili per la funzione di callback nel parametro `data`. Se la chiamata non va a buon fine, i dettagli sull'errore vengono forniti nel parametro `error`.

In genere il codice all'interno della funzione di callback verifica un errore, che elabora nel caso venga restituito. Se non viene restituito un errore, il codice recupera i dati dalla risposta dal parametro `data`. Il formato di base della funzione di callback è simile al seguente esempio.

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

Nell'esempio precedente, i dettagli dell'errore o dei dati restituiti vengono registrati nella console. Ecco un esempio che mostra una funzione di callback trasferita come parte della chiamata a un metodo su un oggetto di servizio.

```
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
  }
});
```

Accesso agli oggetti di richiesta e risposta

All'interno della funzione di callback, la JavaScript parola chiave `this` si riferisce all'oggetto sottostante per la maggior parte dei servizi. `AWS.Response` Nell'esempio seguente, la proprietà

`httpResponse` di un oggetto `AWS.Response` viene utilizzata all'interno di una funzione di callback per registrare i dati e le intestazioni della risposta non elaborati per facilitare il debug.

```
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
  }
});
```

Inoltre, poiché l'oggetto `AWS.Response` ha una proprietà `Request` che contiene l'oggetto `AWS.Request` inviato dalla chiamata al metodo originale, puoi accedere anche ai dettagli della richiesta effettuata.

Utilizzo di un listener di eventi oggetto di richiesta

Se non crei e trasferisci una funzione di callback anonimo come parametro quando chiami un metodo dell'oggetto di servizio, la chiamata al metodo genera un oggetto `AWS.Request` che deve essere inviato manualmente utilizzando il relativo metodo `send`.

Per elaborare la risposta, devi creare un listener di eventi per l'oggetto `AWS.Request` per registrare una funzione di callback per la chiamata al metodo. L'esempio seguente mostra come creare l'oggetto `AWS.Request` per chiamare un metodo dell'oggetto di servizio e il listener di eventi per l'esito positivo.

```
// 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();
```

Dopo aver chiamato il metodo `send` sull'oggetto `AWS.Request`, viene eseguito il gestore eventi quando l'oggetto di servizio riceve un oggetto `AWS.Response`.

Per ulteriori informazioni sull'`AWS.Request`oggetto, consulta `Class: AWS.Request`l'API Reference. Per ulteriori informazioni sull'`AWS.Response`oggetto, consulta Utilizzo dell'oggetto di risposta o `Class: AWS.Response`consulta l'API Reference.

Concatenazione di più callback

Puoi registrare più callback su qualsiasi oggetto di richiesta. È possibile registrare più callback per diversi eventi o per lo stesso evento. Puoi anche concatenare i callback come mostrato nell'esempio seguente.

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

Eventi di completamento degli oggetti di richiesta

L'oggetto `AWS.Request` genera questi eventi di completamento in base alla risposta di ciascun metodo dell'operazione di servizio:

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

Puoi registrarsi una funzione di callback in risposta a uno qualsiasi di questi eventi. Per un elenco completo di tutti gli eventi relativi all'oggetto della richiesta, `Class: AWS.Request`consulta l'API Reference.

Evento success

L'evento `success` viene generato in seguito a una risposta positiva ricevuta dall'oggetto di servizio. Ecco come si registra una funzione di callback per questo evento.

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

La risposta fornisce una proprietà `data` che contiene i dati di risposta serializzati dal servizio. Ad esempio, la seguente chiamata al `listBuckets` metodo dell'oggetto di servizio Amazon S3

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

restituisce la risposta e quindi stampa i seguenti contenuti della proprietà `data` nella console.

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

Evento error

L'evento `error` viene generato in seguito a una risposta di errore ricevuta dall'oggetto di servizio. Ecco come si registra una funzione di callback per questo evento.

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

Quando viene generato l'evento `error`, il valore della proprietà `data` della risposta è `null` e la proprietà `error` contiene i dati dell'errore. L'oggetto `error` associato viene trasferito come primo parametro alla funzione di callback registrata. Ad esempio, il seguente codice:

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

```
}).send();
```

restituisce l'errore e quindi stampa i seguenti dati dell'errore nella console.

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

Evento complete

L'evento `complete` viene generato al termine della chiamata a un oggetto di servizio, indipendentemente dal fatto che la chiamata abbia esito positivo o negativo. Ecco come si registra una funzione di callback per questo evento.

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

Utilizza il callback dell'evento `complete` per gestire qualsiasi pulizia delle richieste che deve essere eseguita indipendentemente dall'esito positivo o negativo. Se utilizzi i dati di risposta all'interno di un callback per l'evento `complete`, verifica le proprietà `response.data` o `response.error` prima di tentare di accedervi, come mostrato nell'esempio seguente.

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

Eventi HTTP degli oggetti di richiesta

L'oggetto `AWS.Request` genera questi eventi HTTP in base alla risposta di ciascun metodo dell'operazione di servizio:

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

- `httpDone`

Puoi registrarsi una funzione di callback in risposta a uno qualsiasi di questi eventi. Per un elenco completo di tutti gli eventi relativi all'oggetto della richiesta, consulta `Class: AWS.Request`l'API Reference.

Evento httpHeaders

L'evento `httpHeaders` viene generato quando vengono inviate le intestazioni dal server remoto. Ecco come si registra una funzione di callback per questo evento.

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

Il parametro `statusCode` della funzione di callback è il codice di stato HTTP. Il parametro `headers` contiene le intestazioni di risposta.

Evento httpData

L'evento `httpData` viene generato per eseguire lo streaming dei pacchetti di dati di risposta dal servizio. Ecco come si registra una funzione di callback per questo evento.

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

Questo evento viene in genere utilizzato per ricevere risposte di grandi dimensioni in blocchi quando il caricamento dell'intera risposta in memoria non è una soluzione pratica. Questo evento ha un parametro `chunk` aggiuntivo che contiene una parte dei dati effettivi dal server.

Se registri un callback per l'evento `httpData`, la proprietà `data` della risposta contiene l'intero output serializzato per la richiesta. Devi rimuovere il listener `httpData` predefinito se non disponi dell'overhead di analisi e memoria extra per i gestori incorporati.

Gli httpUploadProgress ed httpDownloadProgress eventi

L'evento `httpUploadProgress` viene generato quando la richiesta HTTP ha caricato più dati. Allo stesso modo, l'evento `httpDownloadProgress` viene generato quando la richiesta HTTP ha scaricato più dati. Ecco come si registra una funzione di callback per questi eventi.

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

Il parametro `progress` della funzione di callback contiene un oggetto con i byte caricati e totali della richiesta.

Evento httpError

L'evento `httpError` viene generato quando la richiesta HTTP ha esito negativo. Ecco come si registra una funzione di callback per questo evento.

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

Il parametro `error` della funzione di callback contiene l'errore che è stato generato.

Evento httpDone

L'evento `httpDone` viene generato quando il server termina l'invio di dati. Ecco come si registra una funzione di callback per questo evento.

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

Utilizzo di async/await

È possibile utilizzare lo `async/await` schema nelle chiamate a AWS SDK per JavaScript. La maggior parte delle funzioni che accettano un callback non restituiscono una promessa. Poiché si utilizzano solo `await` funzioni che restituiscono una promessa, per utilizzare lo `async/await` schema è necessario concatenare il `.promise()` metodo alla fine della chiamata e rimuovere il callback.

L'esempio seguente utilizza async/await per elencare tutte le tabelle Amazon DynamoDB in. `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**
>
> Non tutti i browser supportano async/await. Vedi Funzioni asincrone per un elenco di browser supportati. async/await

Usare Promises JavaScript

Il metodo `AWS.Request.promise` fornisce un modo per chiamare un'operazione di servizio e gestire il flusso asincrono invece di utilizzare i callback. In Node.js e negli script del browser, viene restituito un oggetto `AWS.Request` quando viene chiamata un'operazione di servizio senza una funzione di callback. Puoi chiamare il metodo `send` della richiesta per effettuare la chiamata di servizio.

Tuttavia, `AWS.Request.promise` avvia immediatamente la chiamata di servizio e restituisce una promessa soddisfatta con la proprietà `data` della risposta o respinta con la proprietà `error` della risposta.

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

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

// handle promise's fulfilled/rejected states
promise.then(
  function(data) {
```

```
    /* process the data */
  },
  function(error) {
    /* handle the error */
  }
);
```

L'esempio seguente restituisce una promessa soddisfatta con un oggetto `data` o respinta con un oggetto `error`. Utilizzando le promesse, un singolo callback non è responsabile per il rilevamento degli errori. Al contrario, il callback corretto viene chiamato in base all'esito positivo o negativo di una richiesta.

```
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);
});
```

Coordinamento di più promesse

In alcune situazioni, il tuo codice deve effettuare più chiamate asincrone che richiedono un intervento solo quando sono state restituite tutte correttamente. Se gestisci tali chiamate singole al metodo asincrono con le promesse, puoi creare una promessa aggiuntiva che utilizza il metodo `all`. Questo metodo soddisfa questa promessa universale se e quando le promesse che trasferisci al metodo vengono soddisfatte. Alla funzione di callback viene trasferito una matrice dei valori delle promesse trasferite al metodo `all`.

Nell'esempio seguente, una AWS Lambda funzione deve effettuare tre chiamate asincrone ad Amazon DynamoDB, ma può essere completata solo dopo aver soddisfatto le promesse per ogni chiamata.

```
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);
});
```

Supporto del browser e di Node.js per le promesse

Il supporto per JavaScript le promesse native (ECMAScript 2015) dipende dal JavaScript motore e dalla versione in cui viene eseguito il codice. Per determinare il supporto per le JavaScript promesse in ogni ambiente in cui il codice deve essere eseguito, consulta la Tabella ECMAScript di compatibilità su. GitHub

Utilizzo di altre implementazioni di promesse

Oltre all'implementazione nativa di Promise nel ECMAScript 2015, puoi anche utilizzare librerie di promesse di terze parti, tra cui:

- bluebird
- RSVP
- Q

Queste librerie di promesse opzionali possono essere utili se hai bisogno che il codice venga eseguito in ambienti che non supportano l'implementazione nativa di promise nel ECMAScript 5 e nel ECMAScript 2015.

Per utilizzare una libreria di promesse di terze parti, imposta una dipendenza per le promesse sull'SDK chiamando il metodo `setPromisesDependency` dell'oggetto di configurazione globale. Negli script del browser, assicurati di caricare la libreria di promesse di terze parti prima di caricare l'SDK. Nell'esempio seguente, l'SDK è configurato per utilizzare l'implementazione nella libreria di promesse bluebird.

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

Per tornare a utilizzare l'implementazione nativa di promise del JavaScript motore, `setPromisesDependency` richiama, passando un nome di libreria `null` anziché un nome di libreria.

Utilizzo dell'oggetto di risposta

Una volta chiamato, un metodo dell'oggetto di servizio restituisce un oggetto `AWS.Response` trasferendolo alla tua funzione di callback. Accedi ai contenuti della risposta attraverso le proprietà dell'oggetto `AWS.Response`. Esistono due proprietà dell'oggetto `AWS.Response` che utilizzi per accedere ai contenuti della risposta:

- Proprietà `data`
- Proprietà `error`

Quando utilizzi il meccanismo di callback standard, queste due proprietà vengono fornite come parametri sulla funzione di callback anonimo, come illustrato nell'esempio seguente.

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

Accesso ai dati restituiti nell'oggetto di risposta

La proprietà `data` dell'oggetto `AWS.Response` contiene i dati serializzati restituiti dalla richiesta di servizio. Quando la richiesta ha esito positivo, la proprietà `data` include un oggetto contenente una mappa per i dati restituiti. La proprietà `data` può essere null se si verifica un errore.

Ecco un esempio di chiamata al `getItem` metodo di una tabella DynamoDB per recuperare il nome di un file di immagine da utilizzare come parte di un gioco.

```
// 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();
```

Per questo esempio, la tabella DynamoDB è una ricerca di immagini che mostrano i risultati del pull di una slot machine come specificato dai parametri in. `slotParams`

Dopo una chiamata riuscita del `getItem` metodo, la `data` proprietà dell'`AWS.Response`oggetto contiene un `Item` oggetto restituito da DynamoDB. Puoi accedere ai dati restituiti in base al parametro `ProjectionExpression` della richiesta, che in questo caso indica il membro `imageFile` dell'oggetto `Item`. Poiché il membro `imageFile` contiene un valore di stringa, puoi accedere al nome file dell'immagine stessa attraverso il valore del membro figlio `S` di `imageFile`.

Paging tramite dati restituiti

A volte il contenuto della proprietà `data` restituito da una richiesta di servizio si estende su più pagine. Puoi accedere alla successiva pagina di dati chiamando il metodo `response.nextPage`. Questo metodo invia una nuova richiesta. La risposta della richiesta può essere acquisita con un callback o con i listener di errore e di operazione riuscita.

Puoi verificare se i dati restituiti da una richiesta di servizio includono pagine di dati aggiuntive chiamando il metodo `response.hasNextPage`. Questo metodo restituisce un valore booleano per indicare se la chiamata `response.nextPage` restituisce dati aggiuntivi.

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

Accesso alle informazioni sugli errori da un oggetto di risposta

La proprietà `error` dell'oggetto `AWS.Response` contiene i dati dell'errore disponibili in caso di errore di servizio o di trasferimento. L'errore restituito presenta il seguente formato.

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

Se si verifica un errore, il valore della proprietà `data` è `null`. Se gestisci eventi che possono essere in uno stato di errore, controlla sempre se la proprietà `error` è stata impostata prima di tentare di accedere al valore della proprietà `data`.

Accesso all'oggetto di richiesta originario

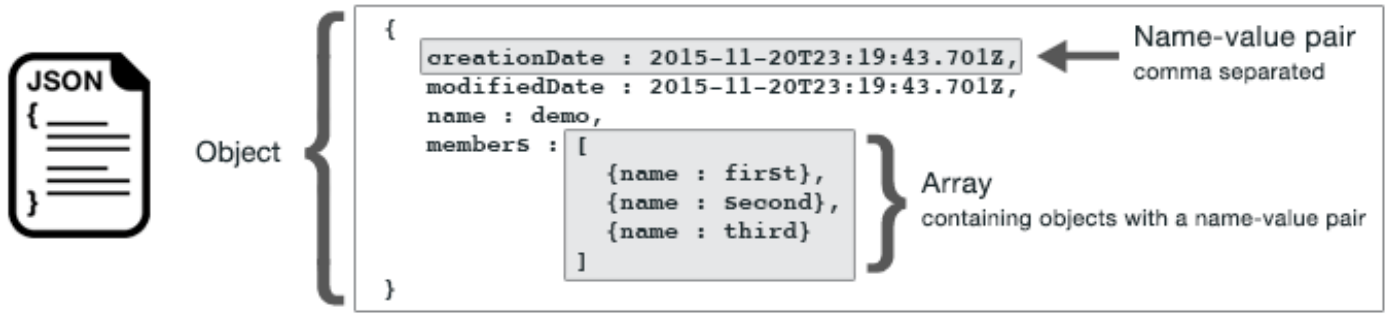
La proprietà `request` fornisce l'accesso all'oggetto `AWS.Request` originario. Può essere utile fare riferimento all'oggetto `AWS.Request` originario per accedere ai parametri originali inviati. Nell'esempio seguente, la proprietà `request` viene utilizzata per accedere al parametro `Key` della richiesta di servizio originale.

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

Utilizzo di JSON

JSON è un formato per lo scambio di dati leggibile sia dall'uomo sia da un computer. Sebbene il nome JSON sia l'acronimo di JavaScript Object Notation, il formato di JSON è indipendente da qualsiasi linguaggio di programmazione.

L'SDK for JavaScript utilizza JSON per inviare dati agli oggetti di servizio quando si effettuano richieste e riceve dati dagli oggetti di servizio come JSON. Per ulteriori informazioni su JSON, consulta json.org.



JSON rappresenta i dati in due modi:

- Un oggetto, una raccolta non ordinata di coppie nome-valore. Un oggetto viene definito all'interno di parentesi graffe sinistra (`{`) e destra (`}`). Ogni coppia nome-valore inizia con il nome, seguita dai due punti e dal valore. Le coppie nome-valore sono separate da virgole.
- Una matrice, che è un insieme ordinato di valori. Una matrice viene definita all'interno di parentesi quadre sinistra (`[`) e destra (`]`). Gli elementi nella matrice sono separati da virgole.

Ecco un esempio di un oggetto JSON che contiene una matrice di oggetti in cui gli oggetti rappresentano le carte in un gioco di carte. Ogni carta è definita da due coppie nome-valore, una che specifica un valore univoco per identificare quella carta e un'altra che specifica un URL che punta all'immagine della carta corrispondente.

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

JSON come parametri dell'oggetto di servizio

Ecco un esempio di JSON semplice utilizzato per definire i parametri di una chiamata a un oggetto del servizio Lambda.

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

L'oggetto `pullParams` è definito da tre coppie nome-valore, separate da virgole e racchiuse fra parentesi graffe sinistra e destra. Quando si forniscono i parametri a una chiamata al metodo dell'oggetto di servizio, i nomi vengono determinati dai nomi dei parametri per il metodo dell'oggetto di servizio che si intende chiamare. Quando si richiama una funzione Lambda`FunctionName`,`InvocationType`, `LogType` e sono i parametri utilizzati per chiamare il metodo su un oggetto `invoke` del servizio Lambda.

Quando si passano parametri a una chiamata al metodo dell'oggetto servizio, fornite l'oggetto JSON alla chiamata al metodo, come illustrato nel seguente esempio di richiamo di una funzione Lambda.

```
lambda = new AWS.Lambda({region: 'us-west-2', apiVersion: '2015-03-31'});
```

```
// create JSON object for service call parameters
var pullParams = {
   FunctionName : 'slotPull',
   InvocationType : 'RequestResponse',
   LogType : 'None'
};
// invoke Lambda function, passing JSON object
lambda.invoke(pullParams, function(err, data) {
   if (err) {
      console.log(err);
   } else {
      console.log(data);
   }
});
```

Restituzione dei dati come JSON

JSON fornisce un modo standard per trasferire i dati tra le parti di un'applicazione che devono inviare più valori allo stesso tempo. I metodi delle classi client nell'API restituiscono in genere JSON nel parametro `data` trasferito alle relative funzioni di callback. Ad esempio, ecco una chiamata al `getBucketCors` metodo della classe client Amazon S3.

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

Il valore di `data` è un oggetto JSON, in questo esempio JSON che descrive la configurazione CORS corrente per un bucket Amazon S3 specificato.

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

```
      }
   ]
}
```

Riprova la strategia nella v2 AWS SDK per JavaScript

Numerosi componenti di una rete, ad esempio server DNS, switch, sistemi di bilanciamento del carico e altri, possono generare errori in qualsiasi fase del ciclo di vita di una richiesta specifica. La tecnica che viene generalmente utilizzata per gestire queste risposte di errore in un ambiente di rete consiste nell'implementare nuovi tentativi nell'applicazione client. Questa tecnica aumenta l'affidabilità dell'applicazione e riduce i costi operativi per lo sviluppatore. AWS SDKs implementa una logica di ripetizione automatica per le tue AWS richieste.

Comportamento esponenziale dei tentativi basato sul backoff

La AWS SDK per JavaScript v2 implementa la logica di riprova utilizzando il backoff esponenziale con jitter completo per un migliore controllo del flusso. L'idea che sottende al backoff esponenziale è di utilizzare attese progressivamente più lunghe tra i tentativi per le risposte di errore consecutive. Il jitter (ritardo randomizzato) viene utilizzato per prevenire collisioni successive.

Test del ritardo tra tentativi in v2

Per testare il ritardo di ripetizione nella v2, il codice in node_ modules/aws-sdk/lib/event _listeners.js è stato aggiornato al valore presente `console.log` nella variabile delay come segue:

```
// delay < 0 is a signal from customBackoff to skip retries
if (willRetry && delay >= 0) {
  resp.error = null;
  console.log('retry delay: ' + delay);
  setTimeout(done, delay);
} else {
  done();
}
```

Ritardi nei tentativi con la configurazione predefinita

Puoi testare il ritardo per qualsiasi operazione sui client SDK AWS. Chiamiamo `listTables` l'operazione su un client DynamoDB utilizzando il seguente codice:

```
import AWS from "aws-sdk";
```

```
const region = "us-east-1";
const client = new AWS.DynamoDB({ region });
await client.listTables({}).promise();
```

Per testare i nuovi tentativi, eseguiamo la simulazione disconnettendo Internet `NetworkingError` dal dispositivo su cui è in esecuzione il codice di test. Puoi anche configurare il proxy per restituire un errore personalizzato.

Durante l'esecuzione del codice, puoi vedere il ritardo tra i tentativi utilizzando il backoff esponenziale con jitter come segue:

```
retry delay: 7.39361151766359
retry delay: 9.0672860785882
retry delay: 134.89340825668168
retry delay: 398.53559817403965
retry delay: 523.8076165896343
retry delay: 1323.8789643058465
```

Poiché retry utilizza il jitter, otterrete valori diversi nell'esecuzione del codice di esempio.

Ritenta i ritardi con una base personalizzata

La AWS SDK per JavaScript v2 consente di passare un numero base personalizzato di millisecondi da utilizzare nel backoff esponenziale per i nuovi tentativi di operazione. Il valore predefinito è 100 ms per tutti i servizi tranne DynamoDB, dove il valore predefinito è 50 ms.

Testiamo i nuovi tentativi con una base personalizzata di 1000 ms nel modo seguente:

```
...
const client = new AWS.DynamoDB({ region, retryDelayOptions: { base: 1000 } });
...
```

Effettuiamo la simulazione `NetworkingError` disconnettendo Internet dal dispositivo su cui è in esecuzione il codice di test. Si può notare che i valori del ritardo tra tentativi sono più alti rispetto all'esecuzione precedente, in cui l'impostazione predefinita era di 50 o 100 ms.

```
retry delay: 356.2841549924913
retry delay: 1183.5216495444615
retry delay: 2266.997988094194
retry delay: 1244.6948354966453
```

```
retry delay: 4200.323030066383
```

Poiché retry utilizza il jitter, otterrete valori diversi nell'esecuzione del codice di esempio.

Riprova i ritardi con un algoritmo di backoff personalizzato

La AWS SDK per JavaScript v2 consente inoltre di passare una funzione di backoff personalizzata che accetta un conteggio dei tentativi e un errore e restituisce la quantità di tempo di ritardo in millisecondi. Se il risultato è un valore negativo diverso da zero, non verranno effettuati ulteriori tentativi.

Testiamo la funzione di backoff personalizzata che utilizza un backoff lineare con un valore base di 200 ms come segue:

```
...
const client = new AWS.DynamoDB({
  region,
  retryDelayOptions: { customBackoff: (count, error) => (count + 1) * 200 },
});
...
```

Effettuiamo la simulazione `NetworkingError` disconnettendo Internet dal dispositivo su cui è in esecuzione il codice di test. Come puoi vedere, i valori del ritardo tra tentativi sono multipli di 200.

```
retry delay: 200
retry delay: 400
retry delay: 600
retry delay: 800
retry delay: 1000
```

SDK per esempi di JavaScript codice

Gli argomenti di questa sezione contengono esempi di come utilizzare AWS SDK per JavaScript i APIs vari servizi per eseguire attività comuni.

Trovate il codice sorgente di questi esempi e di altri nel repository degli esempi di codice di AWS documentazione su GitHub. Per proporre un nuovo esempio di codice da far produrre al team addetto alla AWS documentazione, crea una nuova richiesta. Il team sta cercando di produrre esempi di codice che coprano scenari e casi d'uso più ampi rispetto ai semplici frammenti di codice che coprono solo le singole chiamate API. Per istruzioni, consultate la sezione Codice di creazione nelle Linee guida per i contributi.

Argomenti

CloudWatch Esempi Amazon

Amazon CloudWatch (CloudWatch) è un servizio Web che monitora le risorse e le applicazioni Amazon Web Services su cui esegui AWS in tempo reale. Puoi utilizzarlo CloudWatch per raccogliere e tenere traccia delle metriche, che sono variabili che puoi misurare per le tue risorse e applicazioni. CloudWatch gli allarmi inviano notifiche o apportano automaticamente modifiche alle risorse che stai monitorando in base a regole da te definite.

L' JavaScript API for CloudWatch è esposta tramite le classi `AWS.CloudWatch`AWS.CloudWatchEvents, e `AWS.CloudWatchLogs` client. Per ulteriori informazioni sull'utilizzo delle classi CloudWatch client, consulta `Class: AWS.CloudWatch`Class: `AWS.CloudWatchEvents`, e `Class: AWS.CloudWatchLogs`nel riferimento all'API.

Argomenti

- Creazione di allarmi in Amazon CloudWatch
- Utilizzo delle azioni di allarme in Amazon CloudWatch
- Ottenere metriche da Amazon CloudWatch
- Invio di eventi ad Amazon CloudWatch Events
- Utilizzo dei filtri di abbonamento in Amazon CloudWatch Logs

Creazione di allarmi in Amazon CloudWatch

Questo esempio di codice di Node.js illustra:

- Come recuperare le informazioni di base sui tuoi CloudWatch allarmi.
- Come creare ed eliminare un CloudWatch allarme.

Lo scenario

Un allarme controlla un singolo parametro in un periodo di tempo specificato ed esegue una o più operazioni in base al valore del parametro relativo a una determinata soglia in una serie di periodi di tempo.

In questo esempio, una serie di moduli Node.js vengono utilizzati per creare allarmi in CloudWatch. I moduli Node.js utilizzano l'SDK per JavaScript creare allarmi utilizzando questi metodi della classe `AWS.CloudWatch` client:

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

Per ulteriori informazioni sugli CloudWatch allarmi, consulta Creating Amazon CloudWatch Alarms nella Amazon CloudWatch User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Descrivere gli allarmi

Crea un modulo Node.js con il nome del file `cw_describealarms.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch, crea un oggetto di `AWS.CloudWatch` servizio. Crea un oggetto JSON per contenere i parametri per il recupero delle descrizioni dell'allarme, limitando gli allarmi restituiti a quelli con stato `INSUFFICIENT_DATA`. Quindi, chiama il metodo `describeAlarms` dell'oggetto di servizio `AWS.CloudWatch`.

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

```
// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node cw_describealarms.js
```

Questo codice di esempio può essere trovato qui GitHub.

Creazione di un allarme per una CloudWatch metrica

Crea un modulo Node.js con il nome del file `cw_putmetricalarm.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch, crea un oggetto `AWS.CloudWatch` di servizio. Crea un oggetto JSON per i parametri necessari per creare un allarme basato su una metrica, in questo caso l'utilizzo della CPU di un'istanza Amazon EC2. I parametri rimanenti sono impostati per attivare l'allarme quando il parametro supera una soglia del 70 percento. Quindi, chiama il metodo `describeAlarms` dell'oggetto di servizio `AWS.CloudWatch`.

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

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
```

```
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node cw_putmetricalarm.js
```

Questo codice di esempio può essere trovato qui. GitHub

Eliminare un allarme

Crea un modulo Node.js con il nome del file `cw_deletealarms.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch, crea un oggetto `AWS.CloudWatch` di servizio. Crea un oggetto JSON che contiene i nomi degli allarmi da eliminare. Quindi, chiama il metodo `deleteAlarms` dell'oggetto di servizio `AWS.CloudWatch`.

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

```
// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node cw_deletealarms.js
```

Questo codice di esempio può essere trovato qui GitHub.

Utilizzo delle azioni di allarme in Amazon CloudWatch

Questo esempio di codice di Node.js illustra:

- Come modificare automaticamente lo stato delle istanze Amazon EC2 in base a un allarme. CloudWatch

Lo scenario

Utilizzando le azioni di allarme, puoi creare allarmi che interrompono, terminano, riavviano o ripristinano automaticamente le tue istanze Amazon EC2. Puoi utilizzare le operazioni di arresto o termine quando non è più necessaria l'esecuzione di un'istanza. È possibile utilizzare le azioni di riavvio e ripristino per riavviare automaticamente tali istanze.

In questo esempio, una serie di moduli Node.js viene utilizzata per definire un'azione di allarme CloudWatch che attiva il riavvio di un'istanza Amazon EC2. I moduli Node.js utilizzano l'SDK per JavaScript gestire le istanze Amazon EC2 utilizzando questi metodi della `CloudWatch` classe client:

- `enableAlarmActions`
- `disableAlarmActions`

Per ulteriori informazioni sulle azioni di CloudWatch allarme, consulta Create alarms to stop, terminate, reboot or recovery an instance nella Amazon CloudWatch User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea un ruolo IAM la cui policy conceda l'autorizzazione a descrivere, riavviare, arrestare o terminare un'istanza Amazon EC2. Per ulteriori informazioni sulla creazione di un ruolo IAM, consulta Creating a Role to Delegate Permissions to an AWS Service nella IAM User Guide.

Utilizzare la seguente policy di ruolo quando si crea un ruolo IAM.

JSON

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

```
            "Resource": [
               "*"
            ]
         }
      ]
}
```

Configura l'SDK per JavaScript creando un oggetto di configurazione globale, quindi impostando la regione per il codice. Nell'esempio, la regione è impostata su `us-west-2`.

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

Creare e abilitare operazioni su un allarme

Crea un modulo Node.js con il nome del file `cw_enablealarmactions.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch, crea un oggetto `AWS.CloudWatch` di servizio.

Crea un oggetto JSON per contenere i parametri per la creazione di un allarme, specificando `ActionsEnabled` as `true` e una serie ARNs di azioni che l'allarme attiverà. Chiama il metodo `putMetricAlarm` dell'oggetto di servizio `AWS.CloudWatch`, che crea l'allarme se non esiste o lo aggiorna se è già disponibile.

Nella funzione di callback per, una volta completata con success`putMetricAlarm`, crea un oggetto JSON contenente il nome dell'allarme. CloudWatch Chiama il metodo `enableAlarmActions` per abilitare l'operazione di allarme.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node cw_enablealarmactions.js
```

Questo codice di esempio può essere trovato qui. GitHub

Disattivare le operazioni su un allarme

Crea un modulo Node.js con il nome del file `cw_disablealarmactions.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch, crea un oggetto `AWS.CloudWatch` di servizio. Crea un oggetto JSON contenente il nome dell' CloudWatch allarme. Chiama il metodo `disableAlarmActions` per disabilitare l'operazione per questo allarme.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node cw_disablealarmactions.js
```

Questo codice di esempio può essere trovato qui. GitHub

Ottenere metriche da Amazon CloudWatch

Questo esempio di codice di Node.js illustra:

- Come recuperare un elenco di metriche CloudWatch pubblicate.
- Come pubblicare punti dati nelle metriche. CloudWatch

Lo scenario

I parametri sono dati riguardanti le prestazioni dei sistemi. Puoi abilitare il monitoraggio dettagliato di alcune risorse, come le istanze Amazon EC2, o i parametri delle tue applicazioni.

In questo esempio, una serie di moduli Node.js vengono utilizzati per ottenere metriche da CloudWatch e inviare eventi ad Amazon CloudWatch Events. I moduli Node.js utilizzano l'SDK per JavaScript ottenere metriche dall' CloudWatchutilizzo di questi metodi della `CloudWatch` classe client:

- `listMetrics`
- `putMetricData`

Per ulteriori informazioni sui CloudWatch parametri, consulta Using Amazon CloudWatch Metrics nella Amazon CloudWatch User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Elencazione dei parametri

Crea un modulo Node.js con il nome del file `cw_listmetrics.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch, crea un oggetto di `AWS.CloudWatch` servizio. Crea un oggetto JSON contenente i parametri necessari per elencare i parametri all'interno dello spazio dei nomi `AWS/Logs`. Chiama il metodo `listMetrics` per elencare il parametro `IncomingLogEvents`.

```
// Load the AWS SDK for Node.js
```

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

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node cw_listmetrics.js
```

Questo codice di esempio può essere trovato qui GitHub.

Invio dei parametri personalizzati

Crea un modulo Node.js con il nome del file `cw_putmetricdata.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch, crea un oggetto `AWS.CloudWatch` di servizio. Crea un oggetto JSON contenente i parametri necessari per inviare un punto dati per il parametro personalizzato `PAGES_VISITED`. Chiama il metodo `putMetricData`.

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

```
// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node cw_putmetricdata.js
```

Questo codice di esempio può essere trovato qui GitHub.

Invio di eventi ad Amazon CloudWatch Events

Questo esempio di codice di Node.js illustra:

- Come creare e aggiornare una regola utilizzata per attivare un evento.
- Come definire uno o più target per rispondere a un evento.
- Come inviare eventi che corrispondono ai target per la gestione.

Lo scenario

CloudWatch Events fornisce un flusso quasi in tempo reale di eventi di sistema che descrivono le modifiche nelle risorse di Amazon Web Services a qualsiasi destinazione. Utilizzando semplici regole, puoi abbinare gli eventi e instradarli verso una o più funzioni o flussi target.

In questo esempio, una serie di moduli Node.js vengono utilizzati per inviare CloudWatch eventi a Events. I moduli Node.js utilizzano l'SDK per JavaScript gestire le istanze utilizzando questi metodi della classe `CloudWatchEvents` client:

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

Per ulteriori informazioni sugli CloudWatch eventi, consulta Adding Events with PutEvents nella Amazon CloudWatch Events User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea una funzione Lambda usando il blueprint hello-world come destinazione per gli eventi. Per scoprire come, consulta la Fase 1: Creare una AWS Lambda funzione nella Guida per l'utente di Amazon CloudWatch Events.
- Crea un ruolo IAM la cui politica conceda l'autorizzazione a CloudWatch Events e che lo `events.amazonaws.com` includa come entità affidabile. Per ulteriori informazioni sulla creazione

di un ruolo IAM, consulta Creating a Role to Delegate Permissions to an AWS Service nella IAM User Guide.

Utilizzare la seguente policy di ruolo quando si crea un ruolo IAM.

JSON

```
{
   "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"
      }
   ]
}
```

Utilizzare la seguente relazione di trust quando si crea un ruolo IAM.

JSON

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

```
}
```

Creazione di una regola pianificata

Crea un modulo Node.js con il nome del file `cwe_putrule.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere agli CloudWatch eventi, crea un oggetto di `AWS.CloudWatchEvents` servizio. Crea un oggetto JSON che contiene i parametri necessari per specificare la nuova regola pianificata, che includano i seguenti elementi:

- Un nome per la regola
- L'ARN del ruolo IAM creato in precedenza
- Un'espressione per pianificare l'attivazione della regola ogni cinque minuti

Chiama il metodo `putRule` per creare la regola. La richiamata restituirà l'ARN della regola nuova o aggiornata.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node cwe_putrule.js
```

Questo codice di esempio può essere trovato qui. GitHub

Aggiungere un obiettivo di AWS Lambda funzione

Crea un modulo Node.js con il nome del file `cwe_puttargets.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch agli eventi, create un oggetto `AWS.CloudWatchEvents` di servizio. Crea un oggetto JSON contenente i parametri necessari per specificare la regola a cui desideri collegare il target, incluso l'ARN della funzione Lambda che hai creato. Chiama il metodo `putTargets` dell'oggetto di servizio `AWS.CloudWatchEvents`.

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

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myCloudWatchEventsTarget",
    },
  ],
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node cwe_puttargets.js
```

Questo codice di esempio può essere trovato qui. GitHub

Invio di eventi

Crea un modulo Node.js con il nome del file `cwe_putevents.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere CloudWatch agli eventi, crea un oggetto `AWS.CloudWatchEvents` di servizio. Crea un oggetto JSON che contiene i parametri necessari per l'invio di eventi. Per ogni evento, includi l'origine dell'evento, le ARNs eventuali risorse interessate dall'evento e i dettagli relativi all'evento. Chiama il metodo `putEvents` dell'oggetto di servizio `AWS.CloudWatchEvents`.

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

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

cwevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node cwe_putevents.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Utilizzo dei filtri di abbonamento in Amazon CloudWatch Logs

Questo esempio di codice di Node.js illustra:

- Come creare ed eliminare filtri per gli eventi di registro in CloudWatch Logs.

Lo scenario

Gli abbonamenti forniscono l'accesso a un feed in tempo reale di eventi di log da CloudWatch Logs e lo distribuiscono ad altri servizi, come uno stream Amazon Kinesis AWS Lambda o, per l'elaborazione, l'analisi o il caricamento personalizzati su altri sistemi. Un filtro di sottoscrizione definisce lo schema da utilizzare per filtrare gli eventi di registro inviati alla risorsa. AWS

In questo esempio, una serie di moduli Node.js vengono utilizzati per elencare, creare ed eliminare un filtro di sottoscrizione in CloudWatch Logs. La destinazione degli eventi di registro è una funzione Lambda. I moduli Node.js utilizzano l'SDK per JavaScript gestire i filtri di sottoscrizione utilizzando questi metodi della classe `CloudWatchLogs` client:

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

Per ulteriori informazioni sugli abbonamenti CloudWatch Logs, consulta Elaborazione in tempo reale dei dati di log con abbonamenti nella Amazon CloudWatch Logs User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea una funzione Lambda come destinazione per gli eventi di registro. È necessario utilizzare l'ARN di questa funzione. Per ulteriori informazioni sulla configurazione di una funzione Lambda, consulta Subscription Filters with AWS Lambda nella Amazon CloudWatch Logs User Guide.
- Crea un ruolo IAM la cui policy conceda l'autorizzazione a richiamare la funzione Lambda che hai creato e conceda l'accesso completo ai CloudWatch log o applica la seguente policy al ruolo di esecuzione che crei per la funzione Lambda. Per ulteriori informazioni sulla creazione di un ruolo IAM, consulta Creating a Role to Delegate Permissions to an Service nella IAM User Guide. AWS

Utilizzare la seguente policy di ruolo quando si crea un ruolo IAM.

JSON

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

Descrivere i filtri di sottoscrizione esistenti

Crea un modulo Node.js con il nome del file `cwl_describesubscriptionfilters.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ai CloudWatch log, crea un `AWS.CloudWatchLogs` oggetto di servizio. Crea un oggetto JSON che contiene i parametri necessari per descrivere i filtri esistenti, tra cui il nome del gruppo di log e il numero massimo di filtri che desideri vengano descritti. Chiama il metodo `describeSubscriptionFilters`.

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

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cwl.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node cwl_describesubscriptionfilters.js
```

Questo codice di esempio può essere trovato qui. GitHub

Creazione di un filtro di sottoscrizione

Crea un modulo Node.js con il nome del file `cwl_putsubscriptionfilter.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ai CloudWatch registri, crea un oggetto `AWS.CloudWatchLogs` di servizio. Crea un oggetto JSON contenente i parametri necessari per creare un filtro, incluso l'ARN della funzione Lambda di destinazione, il nome

del filtro, lo schema di stringhe per il filtraggio e il nome del gruppo di log. Chiama il metodo `putSubscriptionFilters`.

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

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node cwl_putsubscriptionfilter.js
```

Questo codice di esempio può essere trovato qui. GitHub

Eliminare un filtro di sottoscrizione

Crea un modulo Node.js con il nome del file `cwl_deletesubscriptionfilters.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ai CloudWatch registri, crea un oggetto `AWS.CloudWatchLogs` di servizio. Crea un oggetto JSON contenente i parametri necessari per eliminare un filtro, inclusi i nomi del filtro e il gruppo di log. Chiama il metodo `deleteSubscriptionFilters`.

```
// Load the AWS SDK for Node.js
```

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

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

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

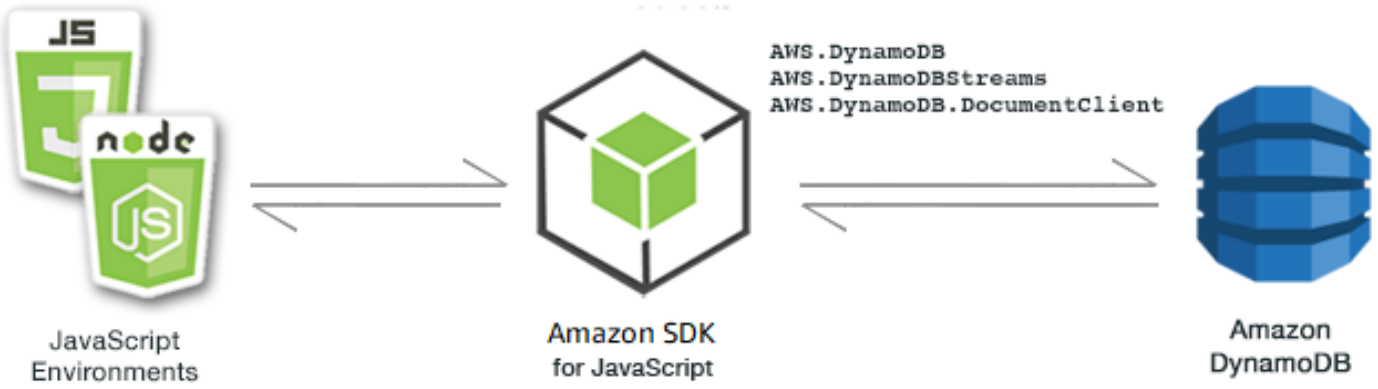
Digita la seguente riga di comando per eseguire l'esempio.

```
node cwl_deletesubscriptionfilter.js
```

Questo codice di esempio può essere trovato qui. GitHub

Esempi di Amazon DynamoDB

Amazon DynamoDB è un database cloud NoSQL completamente gestito che supporta modelli di archiviazione di documenti e chiave-valore. È possibile creare tabelle prive di schema per i dati senza la necessità di allestire o gestire i server di database dedicati.



L' JavaScript API per DynamoDB è esposta tramite `AWS.DynamoDB` le classi`AWS.DynamoDBStreams`, `AWS.DynamoDB.DocumentClient` e client. Per ulteriori informazioni sull'utilizzo delle classi client DynamoDB, `Class: AWS.DynamoDB`vedere `Class: AWS.DynamoDBStreams`, `Class: AWS.DynamoDB.DocumentClient`e nel riferimento API.

Argomenti

- Creazione e utilizzo di tabelle in DynamoDB
- Lettura e scrittura di un singolo elemento in DynamoDB
- Lettura e scrittura di elementi in Batch in DynamoDB
- Interrogazione e scansione di una tabella DynamoDB
- Utilizzo del DynamoDB Document Client

Creazione e utilizzo di tabelle in DynamoDB

Questo esempio di codice di Node.js illustra:

- Come creare e gestire tabelle utilizzate per archiviare e recuperare dati da DynamoDB.

Lo scenario

Analogamente ad altri sistemi di database, DynamoDB archivia i dati in tabelle. Una tabella DynamoDB è una raccolta di dati organizzata in elementi analoghi alle righe. Per archiviare o accedere ai dati in DynamoDB, devi creare e lavorare con le tabelle.

In questo esempio, si utilizza una serie di moduli Node.js per eseguire operazioni di base con una tabella DynamoDB. Il codice utilizza l'SDK per JavaScript creare e lavorare con le tabelle utilizzando questi metodi della `AWS.DynamoDB` classe client:

- createTable
- listTables
- describeTable
- deleteTable

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Creazione di una tabella

Crea un modulo Node.js con il nome del file `ddb_createtable.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per creare una tabella, che in questo esempio include il nome e il tipo di dati per ogni attributo, lo schema chiave, il nome della tabella e le unità di throughput sui cui effettuare il provisioning. Chiama il `createTable` metodo dell'oggetto servizio DynamoDB.

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

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
```

```
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_createtable.js
```

Questo codice di esempio può essere trovato qui. GitHub

Elencare le tabelle

Crea un modulo Node.js con il nome del file `ddb_listtables.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Creare un oggetto JSON che contiene i parametri necessari per elencare le tabelle, che in questo esempio limita il numero di tabelle elencate a 10. Chiama il `listTables` metodo dell'oggetto servizio DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_listtables.js
```

Questo codice di esempio può essere trovato qui. GitHub

Descrizione di una tabella

Crea un modulo Node.js con il nome del file `ddb_describetable.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per descrivere una tabella, che in questo esempio include il nome della tabella fornito come parametro della riga di comando. Chiama il `describeTable` metodo dell'oggetto servizio DynamoDB.

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

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};
```

```
// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_describetable.js TABLE_NAME
```

Questo codice di esempio può essere trovato qui. GitHub

Eliminazione di una tabella

Crea un modulo Node.js con il nome del file `ddb_deletetable.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per eliminare una tabella, che in questo esempio include il nome della tabella fornito come parametro della riga di comando. Chiama il `deleteTable` metodo dell'oggetto servizio DynamoDB.

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

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
```

```
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_deletetable.js TABLE_NAME
```

Questo codice di esempio può essere trovato qui. GitHub

Lettura e scrittura di un singolo elemento in DynamoDB

Questo esempio di codice di Node.js illustra:

- Come aggiungere un elemento in una tabella DynamoDB.
- Come recuperare un elemento in una tabella DynamoDB.
- Come eliminare un elemento in una tabella DynamoDB.

Lo scenario

In questo esempio, si utilizza una serie di moduli Node.js per leggere e scrivere un elemento in una tabella DynamoDB utilizzando questi metodi della `AWS.DynamoDB` classe client:

- putItem
- getItem
- deleteItem

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB, vedere. Creazione e utilizzo di tabelle in DynamoDB

Scrittura di un elemento

Crea un modulo Node.js con il nome del file `ddb_putitem.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per aggiungere una voce, che in questo esempio include il nome della tabella e una mappa che definisce gli attributi da impostare e i valori per ogni attributo. Chiama il `putItem` metodo dell'oggetto servizio DynamoDB.

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

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_putitem.js
```

Questo codice di esempio può essere trovato qui. GitHub

Ottenimento di un item

Crea un modulo Node.js con il nome del file `ddb_getitem.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Per identificare la voce da ottenere, è necessario fornire il valore della chiave primaria per la voce nella tabella. Per impostazione predefinita, il metodo `getItem` restituisce tutti i valori degli attributi definiti per la voce. Per ottenere solo un sottoinsieme di tutti i possibili valori degli attributi, specifica un'espressione di proiezione.

Crea un oggetto JSON che contiene i parametri necessari per ottenere una voce, che in questo esempio include il nome della tabella, il nome e il valore della chiave per la voce che si cerca di ottenere e un'espressione di proiezione che identifica l'attributo della voce che si desidera recuperare. Chiama il `getItem` metodo dell'oggetto servizio DynamoDB.

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

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
```

```
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_getitem.js
```

Questo codice di esempio può essere trovato qui. GitHub

Eliminazione di un item

Crea un modulo Node.js con il nome del file ddb_`deleteitem.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per eliminare una voce, che in questo esempio include il nome della tabella e il nome e il valore della chiave della voce che si sta eliminando. Chiama il `deleteItem` metodo dell'oggetto servizio DynamoDB.

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

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_deleteitem.js
```

Questo codice di esempio può essere trovato qui. GitHub

Lettura e scrittura di elementi in Batch in DynamoDB

Questo esempio di codice di Node.js illustra:

- Come leggere e scrivere batch di elementi in una tabella DynamoDB.

Lo scenario

In questo esempio, si utilizza una serie di moduli Node.js per inserire un batch di elementi in una tabella DynamoDB e leggere un batch di elementi. Il codice utilizza l'SDK per JavaScript eseguire operazioni di lettura e scrittura in batch utilizzando questi metodi della classe client DynamoDB:

- batchGetItem
- batchWriteItem

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB, vedere. Creazione e utilizzo di tabelle in DynamoDB

Lettura delle voci in batch

Crea un modulo Node.js con il nome del file `ddb_batchgetitem.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per ottenere un batch di voci, che in questo esempio include il nome di una o più tabelle da cui leggere, i valori delle chiavi da leggere in ciascuna tabella e l'espressione di proiezione che specifica gli attributi da restituire. Chiama il `batchGetItem` metodo dell'oggetto servizio DynamoDB.

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

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_batchgetitem.js
```

Questo codice di esempio può essere trovato qui. GitHub

Scrittura delle voci in batch

Crea un modulo Node.js con il nome del file ddb_batchwriteitem.js. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un AWS.DynamoDB oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per ottenere un batch di voci, che in questo esempio include la tabella in cui si desidera scrivere le voci, la chiave che si desidera scrivere per ciascuna voce e gli attributi con i relativi valori. Chiama il batchWriteItem metodo dell'oggetto servizio DynamoDB.

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

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
```

```
    ],
  },
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_batchwriteitem.js
```

Questo codice di esempio può essere trovato qui. GitHub

Interrogazione e scansione di una tabella DynamoDB

Questo esempio di codice di Node.js illustra:

- Come interrogare e scansionare una tabella DynamoDB alla ricerca di elementi.

Lo scenario

Eseguire le query consente di trovare le voci in una tabella o un indice secondario utilizzando solo i valori degli attributi della chiave primaria. È necessario fornire un nome e un valore della chiave di partizione da cercare. Puoi inoltre fornire un nome e un valore della chiave di ordinamento e utilizzare un operatore di confronto per perfezionare i risultati della ricerca. La scansione permette di trovare le voci controllando ogni voce nella tabella specificata.

In questo esempio, si utilizza una serie di moduli Node.js per identificare uno o più elementi che si desidera recuperare da una tabella DynamoDB. Il codice utilizza l'SDK per JavaScript interrogare e scansionare le tabelle utilizzando questi metodi della classe client DynamoDB:

- query
- scan

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB, vedere. Creazione e utilizzo di tabelle in DynamoDB

Esecuzione di query su una tabella

In questo esempio si esegue una query su una tabella che contiene informazioni sull'episodio di una serie di video e che restituisce i titoli e sottotitoli degli episodi della seconda stagione dopo l'episodio 9 che contengono una determinata frase nel loro sottotitolo.

Crea un modulo Node.js con il nome del file `ddb_query.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per eseguire una query sulla tabella, che in questo esempio include il nome della tabella, i `ExpressionAttributeValues` necessari per la query, una `KeyConditionExpression` che utilizza tali valori per definire quali voci la query restituisce e i nomi dei valori degli attributi da restituire per ciascuna voce. Chiama il `query` metodo dell'oggetto servizio DynamoDB.

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

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
```

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

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_query.js
```

Questo codice di esempio può essere trovato qui. GitHub

Scansione di una tabella

Crea un modulo Node.js con il nome del file `ddb_scan.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un `AWS.DynamoDB` oggetto servizio. Crea un oggetto JSON che contiene i parametri necessari per eseguire la scansione della tabella per le voci, che in questo esempio include il nome della tabella, l'elenco dei valori degli attributi da restituire per ogni corrispondenza e un'espressione per filtrare il set di risultati per trovare le voci che contengono una determinata frase. Chiama il `scan` metodo dell'oggetto servizio DynamoDB.

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

```
// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddb_scan.js
```

Questo codice di esempio può essere trovato qui. GitHub

Utilizzo del DynamoDB Document Client

Questo esempio di codice di Node.js illustra:

- Come accedere a una tabella DynamoDB utilizzando il client di documenti.

Lo scenario

Il client di documenti DynamoDB semplifica l'utilizzo degli elementi astraendo la nozione di valori degli attributi. Questa astrazione annota i JavaScript tipi nativi forniti come parametri di input e converte i dati di risposta annotati in tipi nativi. JavaScript

Per ulteriori informazioni sulla classe DynamoDB Document Client, `AWS.DynamoDB.DocumentClient`consulta l'API Reference. Per ulteriori informazioni sulla programmazione con Amazon DynamoDB, consulta Programming with DynamoDB nella Amazon DynamoDB Developer Guide.

In questo esempio, si utilizza una serie di moduli Node.js per eseguire operazioni di base su una tabella DynamoDB utilizzando il client di documenti. Il codice utilizza l'SDK per interrogare e JavaScript scansionare le tabelle utilizzando questi metodi della classe DynamoDB Document Client:

- get
- put
- update
- query
- delete

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB utilizzando l'SDK JavaScript per, vedere. Creazione e utilizzo di tabelle in DynamoDB Puoi anche utilizzare la console DynamoDB per creare una tabella.

Ottenere un item da una tabella.

Crea un modulo Node.js con il nome del file `ddbdoc_get.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un oggetto. `AWS.DynamoDB.DocumentClient` Crea un oggetto JSON che contiene i parametri necessari per ottenere una voce dalla tabella, che in questo esempio include il nome della tabella, il nome della chiave hash nella tabella e il valore della chiave hash della voce che si desidera individuare. Chiama il `get` metodo del client di documenti DynamoDB.

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

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddbdoc_get.js
```

Questo codice di esempio può essere trovato qui. GitHub

Inserimento di una voce in una tabella

Crea un modulo Node.js con il nome del file `ddbdoc_put.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un oggetto. `AWS.DynamoDB.DocumentClient` Crea un oggetto JSON che contiene i parametri necessari per scrivere una voce per la tabella, che in questo esempio include il nome della tabella e una descrizione della voce da aggiungere o aggiornare che include il valore e la chiave hash e i nomi e i valori degli attributi da impostare sull'oggetto. Chiama il `put` metodo del client di documenti DynamoDB.

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

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddbdoc_put.js
```

Questo codice di esempio può essere trovato qui. GitHub

Aggiornamento di una voce in una tabella

Crea un modulo Node.js con il nome del file `ddbdoc_update.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un oggetto. `AWS.DynamoDB.DocumentClient` Crea un oggetto JSON che contiene i parametri necessari per scrivere una voce per la tabella, che in questo esempio include il nome della tabella, la chiave della voce da aggiornare, un set di `UpdateExpressions` che definiscono gli attributi della voce da aggiornare con i token a cui sono stati assegnati i valori nei parametri `ExpressionAttributeValues`. Chiama il `update` metodo del client di documenti DynamoDB.

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

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

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

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

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

```
  } else {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddbdoc_update.js
```

Questo codice di esempio può essere trovato qui. GitHub

Esecuzione di query su una tabella

In questo esempio si esegue una query su una tabella che contiene informazioni sull'episodio di una serie di video e che restituisce i titoli e sottotitoli degli episodi della seconda stagione dopo l'episodio 9 che contengono una determinata frase nel loro sottotitolo.

Crea un modulo Node.js con il nome del file `ddbdoc_query.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un oggetto. `AWS.DynamoDB.DocumentClient` Crea un oggetto JSON che contiene i parametri necessari per eseguire una query sulla tabella, che in questo esempio include il nome della tabella, i `ExpressionAttributeValues` necessari per la query e una `KeyConditionExpression` che utilizza tali valori per definire quali voci la query restituisce, Chiama il `query` metodo del client di documenti DynamoDB.

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

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
```

```
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddbdoc_query.js
```

Questo codice di esempio può essere trovato qui. GitHub

Eliminazione di una voce da una tabella.

Crea un modulo Node.js con il nome del file `ddbdoc_delete.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a DynamoDB, crea un oggetto. `AWS.DynamoDB.DocumentClient` Crea un oggetto JSON che contiene i parametri necessari per eliminare una voce nella tabella, che in questo esempio include il nome della tabella e il nome e il valore della chiave hash della voce che si desidera eliminare. Chiama il `delete` metodo del client di documenti DynamoDB.

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

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};
```

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ddbdoc_delete.js
```

Questo codice di esempio può essere trovato qui. GitHub

EC2 Esempi Amazon

Amazon Elastic Compute Cloud (Amazon EC2) è un servizio Web che fornisce hosting di server virtuali nel cloud. È concepito per rendere più semplice il cloud computing a livello web per gli sviluppatori fornendo il ridimensionamento della capacità di elaborazione.

L' JavaScript API per Amazon EC2 è esposta tramite la classe `AWS.EC2` client. Per ulteriori informazioni sull'utilizzo della classe EC2 client Amazon, `Class: AWS.EC2`consulta il riferimento all'API.

Argomenti

- Creazione di un' EC2 istanza Amazon
- Gestione delle EC2 istanze Amazon

- Lavorare con Amazon EC2 Key Pairs
- Utilizzo di regioni e zone di disponibilità con Amazon EC2
- Lavorare con i gruppi di sicurezza in Amazon EC2
- Utilizzo di indirizzi IP elastici in Amazon EC2

Creazione di un' EC2 istanza Amazon

Questo esempio di codice di Node.js illustra:

- Come creare un' EC2 istanza Amazon da un'Amazon Machine Image (AMI) pubblica.
- Come creare e assegnare tag alla nuova EC2 istanza Amazon.

Informazioni sull'esempio

In questo esempio, utilizzi un modulo Node.js per creare un' EC2 istanza Amazon e assegnarle sia una coppia di key pair che dei tag. Il codice utilizza l'SDK per JavaScript creare e contrassegnare un'istanza utilizzando questi metodi della classe EC2 client Amazon:

- `runInstances`
- `createTags`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività.

- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea una coppia di chiavi. Per informazioni dettagliate, consultare Lavorare con Amazon EC2 Key Pairs. Puoi utilizzare il nome della coppia di chiavi di questo esempio.

Creazione e tagging di un'istanza

Crea un modulo Node.js con il nome del file `ec2_createinstances.js`. Assicurati di configurare il kit SDK come mostrato in precedenza.

Crea un oggetto per passare i parametri per il `runInstances` metodo di AWS. EC2 classe client, incluso il nome della coppia di chiavi da assegnare e l'ID dell'AMI da eseguire. Per chiamare il `runInstances` metodo, crea una promessa per richiamare un oggetto di EC2 servizio Amazon, passando i parametri. Quindi gestisci la risposta nel callback della promessa.

Il codice aggiunge quindi un `Name` tag a una nuova istanza, che la EC2 console Amazon riconosce e visualizza nel campo Nome dell'elenco delle istanze. È possibile aggiungere fino a un massimo di 50 tag per un'istanza, che possono essere aggiunti in una singola chiamata al metodo `createTags`.

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

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

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

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

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

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_createinstances.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Gestione delle EC2 istanze Amazon

Questo esempio di codice di Node.js illustra:

- Come recuperare informazioni di base sulle tue EC2 istanze Amazon.

- Come avviare e interrompere il monitoraggio dettagliato di un' EC2 istanza Amazon.
- Come avviare e interrompere un' EC2 istanza Amazon.
- Come riavviare un' EC2 istanza Amazon.

Lo scenario

In questo esempio, si utilizza una serie di moduli di Node.js per eseguire diverse operazioni di base della gestione dell'istanza. I moduli Node.js utilizzano l'SDK per JavaScript gestire le istanze utilizzando questi metodi della classe EC2 client Amazon:

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

Per ulteriori informazioni sul ciclo di vita delle EC2 istanze Amazon, consulta Instance Lifecycle nella Amazon User Guide. EC2

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea un' EC2 istanza Amazon. Per ulteriori informazioni sulla creazione di EC2 istanze Amazon, consulta Amazon EC2 Instances nella Amazon EC2 User Guide o Amazon EC2 Instances nella Amazon EC2 User Guide.

Descrizione delle istanze

Crea un modulo Node.js con il nome del file `ec2_describeinstances.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto

`AWS.EC2` di servizio. Chiama il `describeInstances` metodo dell'oggetto di EC2 servizio Amazon per recuperare una descrizione dettagliata delle tue istanze.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_describeinstances.js
```

Questo codice di esempio può essere trovato qui. GitHub

Gestione del monitoraggio delle istanze

Crea un modulo Node.js con il nome del file `ec2_monitorinstances.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Aggiungi l'istanza IDs delle istanze per le quali desideri controllare il monitoraggio.

In base al valore di un argomento della riga di comando (`0No0FF`), chiama il `monitorInstances` metodo dell'oggetto di EC2 servizio Amazon per iniziare il monitoraggio dettagliato delle istanze specificate oppure chiama il metodo. `unmonitorInstances` Utilizza il parametro `DryRun` per testare se si dispone dell'autorizzazione per modificare il monitoraggio dell'istanza prima di modificare il monitoraggio di queste istanze.

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

```
    }
  });
}
```

Per eseguire l'esempio, digita la seguente stringa nella riga di comando, specificando `ON` per avviare il monitoraggio dettagliato o `OFF` per interrompere il monitoraggio.

```
node ec2_monitorinstances.js ON
```

Questo codice di esempio può essere trovato qui. GitHub

Arresto a avvio delle istanze

Crea un modulo Node.js con il nome del file `ec2_startstopinstances.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Aggiungi l'istanza IDs delle istanze che desideri avviare o interrompere.

In base al valore di un argomento della riga di comando (`STARToSTOP`), chiama il `startInstances` metodo dell'oggetto di EC2 servizio Amazon per avviare le istanze specificate o il `stopInstances` metodo per interromperle. Utilizza il parametro `DryRun` per verificare se disponi dell'autorizzazione prima di avviare e arrestare effettivamente le istanze selezionate.

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

Per eseguire l'esempio, digita la seguente stringa nella riga di comando, specificando `START` per avviare le istanze o `STOP` per arrestarle.

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

Questo codice di esempio può essere trovato qui. GitHub

Riavvio delle istanze

Crea un modulo Node.js con il nome del file `ec2_rebootinstances.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto EC2 di servizio Amazon. Aggiungi l'istanza IDs delle istanze che desideri riavviare. Chiama il metodo `rebootInstances` dell'oggetto di servizio `AWS.EC2` per riavviare le istanze specificate. Utilizza il parametro `DryRun` per verificare se disponi dell'autorizzazione per riavviare le istanze prima di riavviarle effettivamente

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_rebootinstances.js
```

Questo codice di esempio può essere trovato qui. GitHub

Lavorare con Amazon EC2 Key Pairs

Questo esempio di codice di Node.js illustra:

- Come recuperare le informazioni sulle coppie di chiavi.
- Come creare una key pair per accedere a un' EC2 istanza Amazon.
- Come eliminare una coppia di chiavi esistente.

Lo scenario

Amazon EC2 utilizza la crittografia a chiave pubblica per crittografare e decrittografare le informazioni di accesso. La crittografia a chiave pubblica utilizza una chiave pubblica per crittografare i dati, quindi il destinatario utilizza una chiave privata per decrittografare i dati. La chiave pubblica e quella privata sono note come coppia di chiavi.

In questo esempio, utilizzi una serie di moduli Node.js per eseguire diverse operazioni di gestione di Amazon EC2 key pair. I moduli Node.js utilizzano l'SDK per JavaScript gestire le istanze utilizzando questi metodi della classe EC2 client Amazon:

- `createKeyPair`
- `deleteKeyPair`
- `describeKeyPairs`

Per ulteriori informazioni sulle coppie di EC2 chiavi Amazon, consulta Amazon EC2 Key Pairs nella Amazon EC2 User Guide o Amazon EC2 Key Pairs e Windows Instances nella Amazon EC2 User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Descrizione delle coppie di chiavi

Crea un modulo Node.js con il nome del file `ec2_describekeypairs.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea un oggetto JSON vuoto per contenere i parametri necessari per il metodo `describeKeyPairs` per restituire le descrizioni per tutte le coppie di chiavi. È anche possibile fornire un array di nomi di coppie di chiavi nella porzione `KeyName` dei parametri nel file JSON al metodo `describeKeyPairs`.

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

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

// Retrieve key pair descriptions; no params needed
ec2.describeKeyPairs(function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.KeyPairs));
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_describekeypairs.js
```

Questo codice di esempio può essere trovato qui GitHub.

Creazione di una coppia di chiavi

Ogni coppia di chiavi richiede un nome. Amazon EC2 associa la chiave pubblica al nome specificato come nome della chiave. Crea un modulo Node.js con il nome del file `ec2_createkeypair.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea i parametri JSON per specificare il nome della coppia di chiavi, quindi passali per chiamare il metodo `createKeyPair`.

```
// Load the AWS SDK for Node.js
```

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

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

var params = {
  KeyName: "KEY_PAIR_NAME",
};

// Create the key pair
ec2.createKeyPair(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log(JSON.stringify(data));
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_createkeypair.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di una coppia di chiavi

Crea un modulo Node.js con il nome del file `ec2_deletekeypair.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea i parametri JSON per specificare il nome della coppia di chiavi che vuoi eliminare. Quindi chiama il metodo `deleteKeyPair`.

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

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

var params = {
```

```
  KeyName: "KEY_PAIR_NAME",
};

// Delete the key pair
ec2.deleteKeyPair(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Key Pair Deleted");
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_deletekeypair.js
```

Questo codice di esempio può essere trovato qui GitHub.

Utilizzo di regioni e zone di disponibilità con Amazon EC2

Questo esempio di codice di Node.js illustra:

- Come recuperare le descrizioni per le regioni e le zone di disponibilità.

Lo scenario

Amazon EC2 è ospitato in diverse località in tutto il mondo. Tali località sono composte da regioni e zone di disponibilità. Ciascuna regione è un'area geografica distinta. Ciascuna regione presenta più località isolate, conosciute come zone di disponibilità. Amazon EC2 offre la possibilità di collocare istanze e dati in più posizioni.

In questo esempio, si utilizza una serie di moduli di Node.js per recuperare i dettagli sulle regioni e sulle zone di disponibilità. I moduli Node.js utilizzano l'SDK per JavaScript gestire le istanze utilizzando i seguenti metodi della classe EC2 client Amazon:

- `describeAvailabilityZones`

- `describeRegions`

Per ulteriori informazioni su regioni e zone di disponibilità, consulta Regioni e zone di disponibilità nella Amazon EC2 User Guide o Regioni e zone di disponibilità nella Amazon EC2 User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Descrizione delle regioni e delle zone di disponibilità

Crea un modulo Node.js con il nome del file `ec2_describeregionsandzones.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea un oggetto JSON vuoto da passare come parametro, che restituisce tutte le descrizioni disponibili. Quindi chiama i metodi `describeRegions` e `describeAvailabilityZones`.

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

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

var params = {};

// Retrieves all regions/endpoints that work with EC2
ec2.describeRegions(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Regions: ", data.Regions);
  }
});
```

```
// Retrieves availability zones only for region of the ec2 service object
ec2.describeAvailabilityZones(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Availability Zones: ", data.AvailabilityZones);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_describeregionsandzones.js
```

Questo codice di esempio può essere trovato qui GitHub.

Lavorare con i gruppi di sicurezza in Amazon EC2

Questo esempio di codice di Node.js illustra:

- Come recuperare le informazioni sui gruppi di sicurezza.
- Come creare un gruppo di sicurezza per accedere a un' EC2 istanza Amazon.
- Come eliminare un gruppo di sicurezza esistente.

Lo scenario

Un gruppo EC2 di sicurezza Amazon funge da firewall virtuale che controlla il traffico per una o più istanze. A ciascun gruppo di sicurezza possono essere aggiunte regole che permettono il traffico da o verso le istanze associate. Si possono modificare le regole per un gruppo di sicurezza in qualunque momento; le nuove regole vengono applicate automaticamente a tutte le istanze associate al gruppo di sicurezza.

In questo esempio, utilizzi una serie di moduli Node.js per eseguire diverse EC2 operazioni Amazon che coinvolgono gruppi di sicurezza. I moduli Node.js utilizzano l'SDK per JavaScript gestire le istanze utilizzando i seguenti metodi della classe EC2 client Amazon:

- `describeSecurityGroups`
- `authorizeSecurityGroupIngress`
- `createSecurityGroup`
- `describeVpcs`
- `deleteSecurityGroup`

Per ulteriori informazioni sui gruppi di EC2 sicurezza Amazon, consulta Amazon EC2 Amazon Security Groups for Linux Instances nella Amazon EC2 User Guide o Amazon EC2 Security Groups for Windows Instances nella Amazon EC2 User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Descrizione dei gruppi di sicurezza

Crea un modulo Node.js con il nome del file `ec2_describesecuritygroups.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea un oggetto JSON da passare come parametri, incluso il gruppo IDs per i gruppi di sicurezza che desideri descrivere. Quindi chiama il `describeSecurityGroups` metodo dell'oggetto di EC2 servizio Amazon.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_describesecuritygroups.js
```

Questo codice di esempio può essere trovato qui GitHub.

Creazione di un gruppo di sicurezza e delle regole

Crea un modulo Node.js con il nome del file `ec2_createsecuritygroup.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea un oggetto JSON per i parametri che specificano il nome del gruppo di sicurezza, una descrizione e l'ID del VPC. Trasferisci i parametri al metodo `createSecurityGroup`.

Dopo aver creato il gruppo di sicurezza, è possibile definire le regole per consentire il traffico in entrata. Crea un oggetto JSON per i parametri che specificano il protocollo IP e le porte in entrata su cui l' EC2 istanza Amazon riceverà il traffico. Trasferisci i parametri al metodo `authorizeSecurityGroupIngress`.

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

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

// Variable to hold a ID of a VPC
var vpc = null;

// Retrieve the ID of a VPC
ec2.describeVpcs(function (err, data) {
```

```
  if (err) {
    console.log("Cannot retrieve a VPC", err);
  } else {
    vpc = data.Vpcs[0].VpcId;
    var paramsSecurityGroup = {
      Description: "DESCRIPTION",
      GroupName: "SECURITY_GROUP_NAME",
      VpcId: vpc,
    };
    // Create the instance
    ec2.createSecurityGroup(paramsSecurityGroup, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        var SecurityGroupId = data.GroupId;
        console.log("Success", SecurityGroupId);
        var paramsIngress = {
          GroupId: "SECURITY_GROUP_ID",
          IpPermissions: [
            {
              IpProtocol: "tcp",
              FromPort: 80,
              ToPort: 80,
              IpRanges: [{ CidrIp: "0.0.0.0/0" }],
            },
            {
              IpProtocol: "tcp",
              FromPort: 22,
              ToPort: 22,
              IpRanges: [{ CidrIp: "0.0.0.0/0" }],
            },
          ],
        };
        ec2.authorizeSecurityGroupIngress(paramsIngress, function (err, data) {
          if (err) {
            console.log("Error", err);
          } else {
            console.log("Ingress Successfully Set", data);
          }
        });
      }
    });
  }
```

```
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_createsecuritygroup.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Eliminazione di un gruppo di sicurezza

Crea un modulo Node.js con il nome del file `ec2_deletesecuritygroup.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea i parametri JSON per specificare il nome del gruppo di sicurezza che vuoi eliminare. Quindi chiama il metodo `deleteSecurityGroup`.

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

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

var params = {
  GroupId: "SECURITY_GROUP_ID",
};

// Delete the security group
ec2.deleteSecurityGroup(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Security Group Deleted");
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_deletesecuritygroup.js
```

Questo codice di esempio può essere trovato qui GitHub.

Utilizzo di indirizzi IP elastici in Amazon EC2

Questo esempio di codice di Node.js illustra:

- Come recuperare le descrizioni degli indirizzi IP elastici.
- Come allocare e pubblicare un indirizzo IP elastico.
- Come associare un indirizzo IP elastico a un' EC2 istanza Amazon.

Lo scenario

Un indirizzo IP elastico è un indirizzo IP statico progettato per il cloud computing dinamico. Un indirizzo IP elastico è associato al tuo AWS account. Si tratta di un indirizzo IP pubblico, raggiungibile da Internet. Se l'istanza in uso non dispone di un indirizzo IP pubblico, puoi associare un indirizzo IP elastico all'istanza per abilitare la comunicazione con Internet.

In questo esempio, utilizzi una serie di moduli Node.js per eseguire diverse EC2 operazioni Amazon che coinvolgono indirizzi IP elastici. I moduli Node.js utilizzano l'SDK per JavaScript gestire gli indirizzi IP elastici utilizzando questi metodi della classe EC2 client Amazon:

- `describeAddresses`
- `allocateAddress`
- `associateAddress`
- `releaseAddress`

Per ulteriori informazioni sugli indirizzi IP elastici in Amazon EC2, consulta Elastic IP Addresses nella Amazon EC2 User Guide o Elastic IP Addresses nella Amazon EC2 User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea un' EC2 istanza Amazon. Per ulteriori informazioni sulla creazione di EC2 istanze Amazon, consulta Amazon EC2 Instances nella Amazon EC2 User Guide o Amazon EC2 Instances nella Amazon EC2 User Guide.

Descrizione degli indirizzi IP elastici

Crea un modulo Node.js con il nome del file `ec2_describeaddresses.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea un oggetto JSON da passare come parametro, filtrando gli indirizzi restituiti tra quelli nel VPC. Per recuperare le descrizioni di tutti gli indirizzi IP elastici, ometti un filtro nel JSON dei parametri. Quindi chiama il `describeAddresses` metodo dell'oggetto di EC2 servizio Amazon.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_describeaddresses.js
```

Questo codice di esempio può essere trovato qui GitHub.

Allocazione e associazione di un indirizzo IP elastico a un'istanza Amazon EC2

Crea un modulo Node.js con il nome del file `ec2_allocateaddress.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea un oggetto JSON per i parametri utilizzati per allocare un indirizzo IP elastico, che in questo caso specifica che `Domain` è un VPC. Chiama il `allocateAddress` metodo dell'oggetto di EC2 servizio Amazon.

Se la chiamata riesce, il parametro `data` della funzione di callback ha una proprietà `AllocationId` che identifica l'indirizzo IP elastico allocato.

Crea un oggetto JSON per i parametri utilizzati per associare un indirizzo IP elastico a un' EC2 istanza Amazon, incluso quello `AllocationId` proveniente dall'indirizzo appena assegnato e quello `InstanceId` dell'istanza Amazon EC2 . Quindi chiama il `associateAddresses` metodo dell'oggetto di EC2 servizio Amazon.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_allocateaddress.js
```

Questo codice di esempio può essere trovato qui GitHub.

Rilascio di un indirizzo IP elastico

Crea un modulo Node.js con il nome del file `ec2_releaseaddress.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon EC2, crea un oggetto `AWS.EC2` di servizio. Crea un oggetto JSON per i parametri utilizzati per rilasciare un indirizzo IP elastico, che in questo caso specifica `AllocationId` per l'indirizzo IP elastico. Il rilascio di un indirizzo IP elastico consente inoltre di dissociarlo da qualsiasi istanza Amazon. EC2 Chiama il `releaseAddress` metodo dell'oggetto di EC2 servizio Amazon.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_releaseaddress.js
```

Questo codice di esempio può essere trovato qui GitHub.

AWS Elemental MediaConvert Esempi

AWS Elemental MediaConvert è un servizio di transcodifica video basato su file con funzionalità di livello broadcast. Puoi usarlo per creare risorse per la trasmissione e la distribuzione video-on-demand (VOD) su Internet. Per ulteriori informazioni, consulta la Guida per l'utente AWS Elemental MediaConvert .

L' JavaScript API for MediaConvert è esposta tramite la classe `AWS.MediaConvert` client. Per ulteriori informazioni, `Class: AWS.MediaConvert`consulta il riferimento all'API.

Argomenti

- Creazione e gestione di lavori di transcodifica in MediaConvert
- Utilizzo dei Job Templates in MediaConvert

Creazione e gestione di lavori di transcodifica in MediaConvert

Questo esempio di codice di Node.js illustra:

- Come creare lavori di transcodifica in. MediaConvert
- Come annullare un processo di transcodifica.
- Come recuperare il file JSON per un processo di transcodifica completato.
- Come recuperare un array JSON per un massimo di 20 processi creati più di recente.

Lo scenario

In questo esempio, si utilizza un modulo Node.js per effettuare una chiamata per creare e gestire lavori MediaConvert di transcodifica. A tale scopo, il codice utilizza l' JavaScript SDK utilizzando questi metodi della MediaConvert classe client:

- `createJob`
- `cancelJob`
- `getJob`
- `listJobs`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea e configura bucket Amazon S3 che forniscono storage per i file di input e output dei job. Per i dettagli, consulta Create Storage for Files nella Guida per l'AWS Elemental MediaConvert utente.
- Carica il video di input nel bucket Amazon S3 che hai fornito per lo storage di input. Per un elenco dei codec e contenitori di input video supportati, consulta Codecs e contenitori di input supportati nella Guida per l'utente.AWS Elemental MediaConvert
- Crea un ruolo IAM che dia MediaConvert accesso ai tuoi file di input e ai bucket Amazon S3 in cui sono archiviati i file di output. Per i dettagli, consulta Configurare le autorizzazioni IAM nella Guida per l'AWS Elemental MediaConvert utente.

Definizione di un processo di transcodifica semplice

Crea un modulo Node.js con il nome del file `emc_createjob.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Crea il file JSON che definisce i parametri del processo di transcodifica.

Questi parametri sono dettagliati. È possibile utilizzare la AWS Elemental MediaConvert console per generare i parametri del lavoro JSON scegliendo le impostazioni del processo nella console e quindi

selezionando Mostra lavoro JSON nella parte inferiore della sezione Job. Questo esempio illustra il JSON per un processo semplice.

```
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",
  },
},
```

```
};
```

Creazione di un processo di transcodifica

Dopo aver creato il JSON dei parametri del processo, chiama il metodo `createJob` creando una promessa per chiamare un oggetto di servizio `AWS.MediaConvert` e trasferendo i parametri. Quindi gestisci la risposta nel callback della promessa. L'ID del processo creato viene restituito nei `data` della risposta.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node emc_createjob.js
```

Questo codice di esempio è disponibile qui. GitHub

Annullamento di un processo di transcodifica

Crea un modulo Node.js con il nome del file `emc_canceljob.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Crea il file JSON che include l'ID del processo da annullare. Poi chiama il metodo `cancelJob` creando una promessa per chiamare un oggetto di servizio `AWS.MediaConvert`, trasferendo i parametri. Gestisci la risposta restituita dal callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ec2_canceljob.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Elenco dei processi di transcodifica recenti

Crea un modulo Node.js con il nome del file `emc_listjobs.js`. Assicurati di configurare il kit SDK come mostrato in precedenza.

Crea il JSON dei parametri, inclusi i valori per specificare se ordinare l'elenco in ordine `ASCENDING` o `DESCENDING`, l'ARN della coda dei processi da controllare e lo stato dei processi da includere. Poi chiama il metodo `listJobs` creando una promessa per chiamare un oggetto di servizio `AWS.MediaConvert`, trasferendo i parametri. Gestisci la risposta restituita dal callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node emc_listjobs.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Utilizzo dei Job Templates in MediaConvert

Questo esempio di codice di Node.js illustra:

- Come creare modelli MediaConvert di lavoro.
- Come utilizzare un modello dei processi per creare un processo di transcodifica.
- Come elencare tutti i modelli dei processi.
- Come eliminare i modelli dei processi

Lo scenario

Il codice JSON richiesto per creare un processo di transcodifica in MediaConvert è dettagliato e contiene un gran numero di impostazioni. È possibile semplificare notevolmente la creazione del processo salvando le impostazioni corrette in un modello del processo che è possibile utilizzare per creare processi successivi. In questo esempio, si utilizza un modulo Node.js per chiamare per MediaConvert creare, utilizzare e gestire modelli di lavoro. A tale scopo, il codice utilizza l'SDK utilizzando questi metodi della classe MediaConvert client: JavaScript

- `createJobTemplate`
- `createJob`
- `deleteJobTemplate`
- `listJobTemplates`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni, consultare il sito Web di Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea un ruolo IAM che dia MediaConvert accesso ai tuoi file di input e ai bucket Amazon S3 in cui sono archiviati i file di output. Per i dettagli, consulta Configurare le autorizzazioni IAM nella Guida per l'AWS Elemental MediaConvert utente.

Creazione di un modello del processo

Crea un modulo Node.js con il nome del file `emc_create_jobtemplate.js`. Assicurati di configurare il kit SDK come mostrato in precedenza.

Specifica il file JSON dei parametri per la creazione del modello. È possibile utilizzare la maggior parte dei parametri JSON da un processo di successo precedente per specificare i valori `Settings` nel modello. In questo esempio vengono utilizzate le impostazioni del processo contenute in Creazione e gestione di lavori di transcodifica in MediaConvert.

Chiama il metodo `createJobTemplate` creando una promessa per chiamare un oggetto di servizio `AWS.MediaConvert`, trasferendo i parametri. Quindi gestisci la risposta nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node emc_create_jobtemplate.js
```

Questo codice di esempio può essere trovato qui. GitHub

Creazione di un processo di transcodifica da un modello del processo

Crea un modulo Node.js con il nome del file `emc_template_createjob.js`. Assicurati di configurare il kit SDK come mostrato in precedenza.

Crea il JSON dei parametri di creazione del processo, tra cui il nome del modello del processo e le `Settings` da utilizzare che sono specifiche per il processo. Poi chiama il metodo `createJobs` creando una promessa per chiamare un oggetto di servizio `AWS.MediaConvert`, trasferendo i parametri. Gestisci la risposta restituita dal callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node emc_template_createjob.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Elenco dei modelli dei processi

Crea un modulo Node.js con il nome del file `emc_listtemplates.js`. Assicurati di configurare il kit SDK come mostrato in precedenza.

Crea un oggetto per trasferire i parametri della richiesta vuoti per il metodo `listTemplates` della classe del client `AWS.MediaConvert`. Includi valori per determinare i modelli da elencare (`NAME`, `CREATION DATE`, `SYSTEM`), il numero da elencare e il loro ordinamento. Per chiamare il `listTemplates` metodo, create una promessa di invocazione di un oggetto MediaConvert di servizio, passando i parametri. Quindi gestisci la risposta nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node emc_listtemplates.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione di un modello del processo

Crea un modulo Node.js con il nome del file `emc_deletetemplate.js`. Assicurati di configurare il kit SDK come mostrato in precedenza.

Crea un oggetto per inoltrare il nome del modello del processo da eliminare come parametri per il metodo `deleteJobTemplate` della classe client `AWS.MediaConvert`. Per chiamare il `deleteJobTemplate` metodo, create una promessa di invocazione di un oggetto MediaConvert di servizio, passando i parametri. Gestisci la risposta restituita dal callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node emc_deletetemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

AWS Esempi IAM

AWS Identity and Access Management (IAM) è un servizio Web che consente ai clienti di Amazon Web Services di gestire gli utenti e le autorizzazioni degli utenti in AWS. Il servizio è destinato alle organizzazioni con più utenti o sistemi nel cloud che utilizzano AWS prodotti. Con IAM, puoi gestire centralmente gli utenti, le credenziali di sicurezza come le chiavi di accesso e le autorizzazioni che controllano le AWS risorse a cui gli utenti possono accedere.

L' JavaScript API per IAM è esposta tramite la classe `AWS.IAM` client. Per ulteriori informazioni sull'utilizzo della classe client IAM, `Class: AWS.IAM`consulta il riferimento all'API.

Argomenti

Gestione degli utenti IAM

Questo esempio di codice di Node.js illustra:

- Come recuperare un elenco di utenti IAM.
- Come creare ed eliminare gli utenti.
- Come aggiornare un nome utente

Lo scenario

In questo esempio, una serie di moduli Node.js vengono utilizzati per creare e gestire gli utenti in IAM. I moduli Node.js utilizzano l'SDK per JavaScript creare, eliminare e aggiornare gli utenti utilizzando questi metodi della classe `AWS.IAM` client:

- `createUser`
- `listUsers`
- `updateUser`
- `getUser`
- `deleteUser`

Per ulteriori informazioni sugli utenti IAM, consulta IAM Users nella IAM User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Creazione di un utente

Crea un modulo Node.js con il nome del file `iam_createuser.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari, che include il nome utente da utilizzare per il nuovo utente come parametro della riga di comando.

Chiama il metodo `getUser` dell'oggetto di servizio `AWS.IAM` per vedere se il nome utente esiste già. Se il nome utente non esiste, chiama il metodo `createUser` per crearlo. Se il nome esiste già, scrivi un messaggio a tale riguardo alla console.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
```

```
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_createuser.js USER_NAME
```

Questo codice di esempio può essere trovato qui GitHub.

Elenco degli utenti nell'account

Crea un modulo Node.js con il nome del file `iam_listusers.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per elencare gli utenti, limitando il numero restituito impostando il parametro `MaxItems` su 10. Chiama il metodo `listUsers` dell'oggetto di servizio `AWS.IAM`. Scrivi il primo nome utente e la data di creazione sulla console.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
      console.log("User " + user.UserName + " created", user.CreateDate);
```

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_listusers.js
```

Questo codice di esempio può essere trovato qui GitHub.

Aggiornamento di un nome utente

Crea un modulo Node.js con il nome del file `iam_updateuser.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per elencare gli utenti, specificando sia i nomi utenti attuali sia quelli nuovi come parametro della riga di comando. Chiama il metodo `updateUser` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

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

Per eseguire l'esempio, digita la seguente stringa nella riga di comando, specificando il nome utente attuale seguito da quello nuovo.

```
node iam_updateuser.js ORIGINAL_USERNAME NEW_USERNAME
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di un utente

Crea un modulo Node.js con il nome del file `iam_deleteuser.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari, che include il nome utente da eliminare come parametro della riga di comando.

Chiama il metodo `getUser` dell'oggetto di servizio `AWS.IAM` per vedere se il nome utente esiste già. Se il nome utente non esiste, scrivi un messaggio a tale riguardo alla console. Se l'utente esiste, chiama il metodo `deleteUser` per eliminarlo.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_deleteuser.js USER_NAME
```

Questo codice di esempio può essere trovato qui GitHub.

Lavorare con le policy IAM

Questo esempio di codice di Node.js illustra:

- Come creare ed eliminare le policy IAM.
- Come collegare e scollegare le politiche IAM dai ruoli.

Lo scenario

Concedi le autorizzazioni a un utente creando una policy che è un documento che elenca le operazioni che un utente può eseguire e le risorse che tali operazioni possono influenzare. Qualsiasi operazione o risorsa che non è esplicitamente consentita viene negata come impostazione predefinita. Le policy possono essere create e collegate a utenti, gruppi di utenti, ruoli assunti da utenti e risorse.

In questo esempio, una serie di moduli Node.js vengono utilizzati per gestire le policy in IAM. I moduli Node.js utilizzano l'SDK per JavaScript creare ed eliminare le politiche, nonché per allegare e scollegare le politiche dei ruoli utilizzando questi metodi della `AWS.IAM` classe client:

- `createPolicy`
- `getPolicy`
- `listAttachedRolePolicies`
- `attachRolePolicy`
- `detachRolePolicy`

Per ulteriori informazioni sugli utenti IAM, consulta Overview of Access Management: Permissions and Policies nella IAM User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea un ruolo IAM a cui allegare le policy. Per ulteriori informazioni sulla creazione di ruoli, consulta Creating IAM Roles nella IAM User Guide.

Creazione di una policy IAM

Crea un modulo Node.js con il nome del file `iam_createpolicy.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea due oggetti JSON, uno contenente il documento della policy che desideri creare e l'altro contenente i parametri necessari per creare la policy, che comprende il JSON della policy e il nome che desideri dare alla policy. Assicurati di eseguire la funzione stringify sull'oggetto JSON della policy nei parametri. Chiama il metodo `createPolicy` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
        "dynamodb:DeleteItem",
```

```
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
      ],
      Resource: "RESOURCE_ARN",
    },
  ],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_createpolicy.js
```

Questo codice di esempio può essere trovato qui GitHub.

Ottenere una politica IAM

Crea un modulo Node.js con il nome del file `iam_getpolicy.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per recuperare una policy, che è l'ARN della policy che desideri recuperare. Chiama il metodo `getPolicy` dell'oggetto di servizio `AWS.IAM`. Scrivi la descrizione della policy nella console.

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

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_getpolicy.js
```

Questo codice di esempio può essere trovato qui GitHub.

Collegamento di una policy di ruolo gestita

Crea un modulo Node.js con il nome del file `iam_attachrolepolicy.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON contenente i parametri necessari per ottenere un elenco di policy IAM gestite allegate a un ruolo, che consiste nel nome del ruolo. Fornisci il nome del ruolo come parametro della riga di comando. Chiama il metodo `listAttachedRolePolicies` dell'oggetto di servizio `AWS.IAM`, che restituisce una serie di policy gestite alla funzione di callback.

Controlla i membri della serie per vedere se la policy da collegare al ruolo è già collegata. Se la policy non è collegata, chiama il metodo `attachRolePolicy` per collegarla.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_attachrolepolicy.js IAM_ROLE_NAME
```

Scollegamento di una policy di ruolo gestita

Crea un modulo Node.js con il nome del file `iam_detachrolepolicy.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON contenente i parametri necessari per ottenere un elenco di policy IAM gestite allegate a un ruolo, che consiste nel nome del ruolo. Fornisci il nome del ruolo come parametro della

riga di comando. Chiama il metodo `listAttachedRolePolicies` dell'oggetto di servizio `AWS.IAM`, che restituisce una serie di policy gestite nella funzione di callback.

Controlla i membri della serie per vedere se la policy da scollegare dal ruolo è collegata. Se la policy è collegata, chiama il metodo `detachRolePolicy` per scollegarla.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_detachrolepolicy.js IAM_ROLE_NAME
```

Gestione delle chiavi di accesso IAM

Questo esempio di codice di Node.js illustra:

- Come gestire le chiavi di accesso degli utenti.

Lo scenario

Gli utenti hanno bisogno delle proprie chiavi di accesso per effettuare chiamate programmatiche AWS dall'SDK for. JavaScript Per soddisfare questa esigenza, puoi creare, modificare, visualizzare o ruotare le chiavi di accesso (chiave di accesso IDs e chiavi di accesso segrete) per gli utenti IAM. Come impostazione predefinita, quando una chiave di accesso viene creata, il suo stato è `Active`, il che significa che l'utente può utilizzare la chiave di accesso per le chiamate API.

In questo esempio, vengono utilizzati una serie di moduli Node.js per la gestione delle chiavi di accesso in IAM. I moduli Node.js utilizzano l'SDK per JavaScript gestire le chiavi di accesso IAM utilizzando questi metodi della classe `AWS.IAM` client:

- createAccessKey
- listAccessKeys
- getAccessKeyLastUsed
- updateAccessKey
- deleteAccessKey

Per ulteriori informazioni sulle chiavi di accesso IAM, consulta Access Keys nella IAM User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Creazione delle chiavi di accesso per un utente

Crea un modulo Node.js con il nome del file `iam_createaccesskeys.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON contenente i parametri necessari per creare nuove chiavi di accesso, che include il nome utente IAM. Chiama il metodo `createAccessKey` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio. Assicurati di reindirizzare i dati restituiti a un file di testo per non perdere la chiave segreta, che può essere fornita solo una volta.

```
node iam_createaccesskeys.js > newuserkeys.txt
```

Questo codice di esempio può essere trovato qui. GitHub

Elenco delle chiavi di accesso di un utente

Crea un modulo Node.js con il nome del file `iam_listaccesskeys.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON contenente i parametri necessari per recuperare le chiavi di accesso

dell'utente, che include il nome utente IAM e, facoltativamente, il numero massimo di coppie di chiavi di accesso che desideri elencare. Chiama il metodo `listAccessKeys` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_listaccesskeys.js
```

Questo codice di esempio può essere trovato qui. GitHub

Recupero dell'ultimo utilizzo delle chiavi di accesso

Crea un modulo Node.js con il nome del file `iam_accesskeylastused.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per creare nuove chiavi di accesso, che è l'ID chiave di accesso per cui desideri recuperare le informazioni sull'ultimo utilizzo. Chiama il metodo `getAccessKeyLastUsed` dell'oggetto di servizio `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getAccessKeyLastUsed(
  { AccessKeyId: "ACCESS_KEY_ID" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
    }
  }
);
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_accesskeylastused.js
```

Questo codice di esempio può essere trovato qui GitHub.

Aggiornamento dello stato della chiave di accesso

Crea un modulo Node.js con il nome del file `iam_updateaccesskey.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per aggiornare lo stato di una chiave di accesso, che include l'ID chiave di accesso e lo stato aggiornato. Lo stato può essere `Active` o `Inactive`. Chiama il metodo `updateAccessKey` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
```

```
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_updateaccesskey.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione delle chiavi di accesso

Crea un modulo Node.js con il nome del file `iam_deleteaccesskey.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per eliminare le chiavi di accesso, che include l'ID chiave di accesso e il nome dell'utente. Chiama il metodo `deleteAccessKey` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

iam.deleteAccessKey(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_deleteaccesskey.js
```

Questo codice di esempio può essere trovato qui GitHub.

Utilizzo dei certificati del server IAM

Questo esempio di codice di Node.js illustra:

- Come eseguire attività di base nella gestione dei certificati server per le connessioni HTTPS.

Lo scenario

Per abilitare le connessioni HTTPS al tuo sito Web o alla tua applicazione AWS, è necessario un certificato SSL/TLS del server. Per utilizzare un certificato ottenuto da un provider esterno con il tuo sito Web o la tua applicazione AWS, devi caricare il certificato su IAM o importarlo in AWS Certificate Manager.

In questo esempio, una serie di moduli Node.js vengono utilizzati per gestire i certificati server in IAM. I moduli Node.js utilizzano l'SDK per JavaScript gestire i certificati del server utilizzando questi metodi della classe `AWS.IAM` client:

- listServerCertificates
- getServerCertificate
- updateServerCertificate
- deleteServerCertificate

Per ulteriori informazioni sui certificati server, consulta Working with Server Certificates nella IAM User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Elenco dei certificati server

Crea un modulo Node.js con il nome del file `iam_listservercerts.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Chiama il metodo `listServerCertificates` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_listservercerts.js
```

Questo codice di esempio può essere trovato qui GitHub.

Ottenere un certificato del server

Crea un modulo Node.js con il nome del file `iam_getservercert.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per ottenere un certificato, che include il nome del certificato server che desideri. Chiama il metodo `getServerCertificates` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_getservercert.js
```

Questo codice di esempio può essere trovato qui GitHub.

Aggiornamento di un certificato del server

Crea un modulo Node.js con il nome del file `iam_updateservercert.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per aggiornare un certificato, che include il nome del certificato server esistente, nonché il nome del nuovo certificato. Chiama il metodo `updateServerCertificate` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_updateservercert.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di un certificato del server

Crea un modulo Node.js con il nome del file `iam_deleteservercert.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per eliminare un certificato server, che include il nome del certificato che desideri eliminare. Chiama il metodo `deleteServerCertificates` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
```

```
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_deleteservercert.js
```

Questo codice di esempio può essere trovato qui GitHub.

Gestire gli alias per l'account IAM

Questo esempio di codice di Node.js illustra:

- Come gestire gli alias per l'ID del tuo AWS account.

Lo scenario

Se desideri che l'URL della pagina di accesso contenga il nome della tua azienda o un altro identificativo descrittivo anziché l'ID dell' AWS account, puoi creare un alias per l'ID dell'account. AWS Se crei un alias per l' AWS account, l'URL della pagina di accesso cambia per incorporare l'alias.

In questo esempio, una serie di moduli Node.js vengono utilizzati per creare e gestire gli alias degli account IAM. I moduli Node.js utilizzano l'SDK per JavaScript gestire gli alias utilizzando questi metodi della `AWS.IAM` classe client:

- createAccountAlias
- listAccountAliases
- deleteAccountAlias

Per ulteriori informazioni sugli alias degli account IAM, consulta Your AWS Account ID e Its Alias nella IAM User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Creazione di un alias dell'account

Crea un modulo Node.js con il nome del file `iam_createaccountalias.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto di `AWS.IAM` servizio. Crea un oggetto JSON che contenga i parametri necessari per creare un alias dell'account, che include l'alias che desideri creare. Chiama il metodo `createAccountAlias` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_createaccountalias.js ALIAS
```

Questo codice di esempio può essere trovato qui GitHub.

Elencazione di alias dell'account

Crea un modulo Node.js con il nome del file `iam_listaccountaliases.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per elencare gli alias dell'account, che include il numero massimo di elementi da restituire. Chiama il metodo `listAccountAliases` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_listaccountaliases.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di un alias dell'account

Crea un modulo Node.js con il nome del file `iam_deleteaccountalias.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere a IAM, crea un oggetto `AWS.IAM` di servizio. Crea un oggetto JSON che contenga i parametri necessari per eliminare un alias

dell'account, che include l'alias che desideri eliminare. Chiama il metodo `deleteAccountAlias` dell'oggetto di servizio `AWS.IAM`.

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node iam_deleteaccountalias.js ALIAS
```

Questo codice di esempio può essere trovato qui GitHub.

Esempio di Amazon Kinesis

Amazon Kinesis è una piattaforma per lo streaming di dati AWS, che offre potenti servizi per caricare e analizzare i dati di streaming e offre anche la possibilità di creare applicazioni di dati di streaming personalizzate per esigenze specifiche.

L' JavaScript API per Kinesis è esposta tramite la classe `AWS.Kinesis` client. Per ulteriori informazioni sull'utilizzo della classe client Kinesis, consulta il riferimento `Class: AWS.Kinesis`all'API.

Argomenti

- Registrazione dell'avanzamento dello scorrimento delle pagine Web con Amazon Kinesis

Registrazione dell'avanzamento dello scorrimento delle pagine Web con Amazon Kinesis

Questo script di browser di esempio mostra:

- Come registrare l'avanzamento dello scorrimento in una pagina Web con Amazon Kinesis come esempio di metriche di utilizzo delle pagine in streaming per analisi successive.

Lo scenario

In questo esempio, una semplice pagina HTML simula i contenuti di una pagina di un blog. Mentre il lettore scorre il post simulato sul blog, lo script del browser utilizza l'SDK per JavaScript registrare la distanza di scorrimento verso il basso della pagina e inviare i dati a Kinesis utilizzando il metodo `putRecords`della classe client Kinesis. I dati di streaming acquisiti da Amazon Kinesis Data Streams possono quindi essere elaborati dalle istanze Amazon EC2 e archiviati in uno qualsiasi dei diversi archivi di dati tra cui Amazon DynamoDB e Amazon Redshift.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Crea uno stream Kinesis. È necessario includere l'ARN delle risorse del flusso nello script di browser. Per ulteriori informazioni sulla creazione di Amazon Kinesis Data Streams, consulta Managing Kinesis Streams nella Amazon Kinesis Data Streams Developer Guide.
- Crea un pool di identità Amazon Cognito con accesso abilitato per le identità non autenticate. È necessario includere l'ID del pool di identità nel codice per ottenere le credenziali per lo script di

browser. Per ulteriori informazioni sui pool di identità di Amazon Cognito, consulta Identity Pools nella Amazon Cognito Developer Guide.

- Crea un ruolo IAM la cui policy conceda l'autorizzazione a inviare dati a un flusso Kinesis. Per ulteriori informazioni sulla creazione di un ruolo IAM, consulta Creating a Role to Delegate Permissions to an AWS Service nella IAM User Guide.

Utilizzare la seguente policy di ruolo quando si crea un ruolo IAM.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "mobileanalytics:PutEvents",
            "cognito-sync:*"
         ],
         "Resource": [
            "*"
         ]
      },
      {
         "Effect": "Allow",
         "Action": [
            "kinesis:Put*"
         ],
         "Resource": [
            "arn:aws:kinesis:us-east-1:111122223333:stream/stream-name"
         ]
      }
   ]
}
```

La pagina del blog

Il codice HTML per la pagina del blog consiste principalmente in una serie di paragrafi contenuti in un elemento `<div>`. L'altezza scorribile di questo `<div>` viene utilizzata per aiutare a calcolare la

distanza che un lettore ha scorso attraverso i contenuti durante la lettura. L'HTML contiene anche una coppia di elementi `<script>`. Uno di questi elementi aggiunge l'SDK per JavaScript alla pagina e l'altro aggiunge lo script del browser che registra l'avanzamento dello scorrimento sulla pagina e lo segnala a Kinesis.

```
<!DOCTYPE html>
<html>
    <head>
        <title>AWS SDK for JavaScript - Amazon Kinesis Application</title>
    </head>
    <body>
        <div id="BlogContent" style="width: 60%; height: 800px; overflow: auto;margin:
 auto; text-align: center;">
            <div>
                <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum
 vitae nulla eget nisl bibendum feugiat. Fusce rhoncus felis at ultricies luctus.
 Vivamus fermentum cursus sem at interdum. Proin vel lobortis nulla. Aenean rutrum
 odio in tellus semper rhoncus. Nam eu felis ac augue dapibus laoreet vel in erat.
 Vivamus vitae mollis turpis. Integer sagittis dictum odio. Duis nec sapien diam.
 In imperdiet sem nec ante laoreet, vehicula facilisis sem placerat. Duis ut metus
 egestas, ullamcorper neque et, accumsan quam. Class aptent taciti sociosqu ad litora
 torquent per conubia nostra, per inceptos himenaeos.
                </p>
                <!-- Additional paragraphs in the blog page appear here -->
            </div>
        </div>
        <script src="https://sdk.amazonaws.com/js/aws-sdk-2.283.1.min.js"></script>
        <script src="kinesis-example.js"></script>
    </body>
</html>
```

Configurazione dell'SDK

Ottieni le credenziali necessarie per configurare l'SDK chiamando il `CognitoIdentityCredentials` metodo, fornendo l'ID del pool di identità di Amazon Cognito. In caso di successo, crea l'oggetto di servizio Kinesis nella funzione di callback.

Il seguente frammento di codice mostra questa fase. (Per l'esempio completo, consulta Acquisizione del codice dello Scroll Progress della pagina Web.)

```
// Configure Credentials to use Cognito
```

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

AWS.config.region = "REGION";
// We're going to partition Amazon Kinesis records based on an identity.
// We need to get credentials first, then attach our event listeners.
AWS.config.credentials.get(function (err) {
  // attach event listener
  if (err) {
    alert("Error retrieving credentials.");
    console.error(err);
    return;
  }
  // create Amazon Kinesis service object
  var kinesis = new AWS.Kinesis({
    apiVersion: "2013-12-02",
  });
```

Creazione dei record di scorrimento

Lo scroll progress viene calcolato utilizzando le proprietà `scrollHeight` e `scrollTop` di `<div>` che contengono i contenuti del post del blog. Ogni record di scorrimento viene creato in una funzione di listener di eventi per l'`scroll`evento e quindi aggiunto a una serie di record per l'invio periodico a Kinesis.

Il seguente frammento di codice mostra questa fase. (Per l'esempio completo, consulta Acquisizione del codice dello Scroll Progress della pagina Web.)

```
  // Get the ID of the Web page element.
  var blogContent = document.getElementById("BlogContent");

  // Get Scrollable height
  var scrollableHeight = blogContent.clientHeight;

  var recordData = [];
  var TID = null;
  blogContent.addEventListener("scroll", function (event) {
    clearTimeout(TID);
    // Prevent creating a record while a user is actively scrolling
    TID = setTimeout(function () {
      // calculate percentage
      var scrollableElement = event.target;
```

```
    var scrollHeight = scrollableElement.scrollHeight;
    var scrollTop = scrollableElement.scrollTop;

    var scrollTopPercentage = Math.round((scrollTop / scrollHeight) * 100);
    var scrollBottomPercentage = Math.round(
      ((scrollTop + scrollableHeight) / scrollHeight) * 100
    );

    // Create the Amazon Kinesis record
    var record = {
      Data: JSON.stringify({
        blog: window.location.href,
        scrollTopPercentage: scrollTopPercentage,
        scrollBottomPercentage: scrollBottomPercentage,
        time: new Date(),
      }),
      PartitionKey: "partition-" + AWS.config.credentials.identityId,
    };
    recordData.push(record);
  }, 100);
});
```

Invio di record a Kinesis

Una volta al secondo, se nell'array sono presenti record, tali record in sospeso vengono inviati a Kinesis.

Il seguente frammento di codice mostra questa fase. (Per l'esempio completo, consulta Acquisizione del codice dello Scroll Progress della pagina Web.)

```
// upload data to Amazon Kinesis every second if data exists
setInterval(function () {
  if (!recordData.length) {
    return;
  }
  // upload data to Amazon Kinesis
  kinesis.putRecords(
    {
      Records: recordData,
      StreamName: "NAME_OF_STREAM",
    },
    function (err, data) {
      if (err) {
```

```
          console.error(err);
        }
      }
    );
    // clear record data
    recordData = [];
  }, 1000);
});
```

Acquisizione del codice dello Scroll Progress della pagina Web

Ecco il codice dello script del browser per l'esempio di avanzamento dello scorrimento delle pagine Web di Kinesis Capturing.

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

Esempi di Amazon S3

Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) è un servizio Web che fornisce storage nel cloud altamente scalabile. Amazon S3 offre uno storage di oggetti facile da usare, con una semplice interfaccia di servizi Web per archiviare e recuperare qualsiasi quantità di dati da qualsiasi punto del Web.

L' JavaScript API per Amazon S3 è esposta tramite la classe `AWS.S3` client. Per ulteriori informazioni sull'uso della classe client Amazon S3, consulta il riferimento `Class: AWS.S3`all'API.

Argomenti

- Esempi di browser Amazon S3
- Esempi di Amazon S3 Node.js

Esempi di browser Amazon S3

I seguenti argomenti mostrano due esempi di come AWS SDK per JavaScript può essere utilizzato nel browser per interagire con i bucket Amazon S3.

- Il primo mostra uno scenario semplice in cui le foto esistenti in un bucket Amazon S3 possono essere visualizzate da qualsiasi utente (non autenticato).
- Il secondo mostra uno scenario più complesso in cui gli utenti possono eseguire operazioni sulle foto contenute nel bucket, ad esempio caricarle, eliminarle, ecc.

Argomenti

- Visualizzazione di foto in un bucket Amazon S3 da un browser
- Caricamento di foto su Amazon S3 da un browser

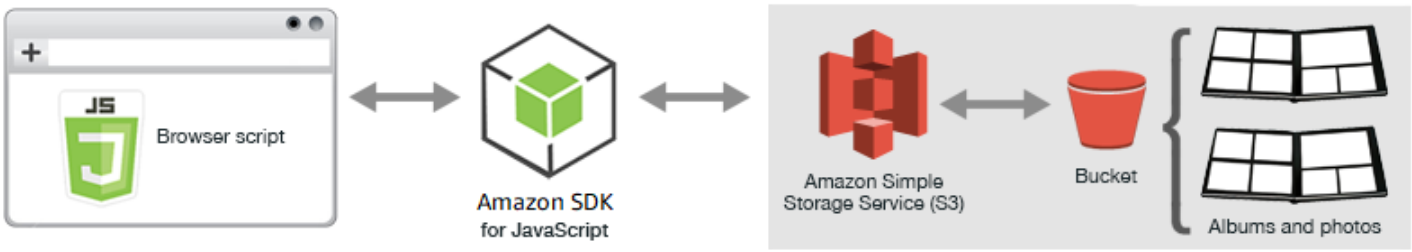
Visualizzazione di foto in un bucket Amazon S3 da un browser

Questo esempio di codice dello script di browser illustra:

- Come creare un album di foto in un bucket Amazon Simple Storage Service (Amazon S3) e consentire agli utenti non autenticati di visualizzare le foto.

Lo scenario

In questo esempio, una semplice pagina HTML fornisce un'applicazione basata su browser per visualizzare le foto in un album di foto. L'album fotografico si trova in un bucket Amazon S3 in cui vengono caricate le foto.



Lo script del browser utilizza l'SDK JavaScript per interagire con un bucket Amazon S3. Lo script utilizza il `listObjects`metodo della classe client Amazon S3 per consentire la visualizzazione degli album fotografici.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività.

> **Note**
>
> In questo esempio, devi utilizzare la stessa AWS regione sia per il bucket Amazon S3 che per il pool di identità Amazon Cognito.

Per creare il bucket

Nella console Amazon S3, crea un bucket Amazon S3 in cui archiviare album e foto. Per ulteriori informazioni sull'utilizzo della console per creare un bucket S3, consulta Creating a Bucket nella Amazon Simple Storage Service User Guide.

Quando crei il bucket S3, assicurati di eseguire le operazioni seguenti:

- Annotare il nome del bucket in modo che sia possibile utilizzarlo in una successiva attività relativa ai prerequisiti, Configura autorizzazioni ruolo.
- Scegli una AWS regione in cui creare il bucket. Questa deve essere la stessa regione che utilizzerai per creare un pool di identità di Amazon Cognito in un'attività prerequisita successiva, Crea un pool di identità.
- Configura le autorizzazioni del bucket seguendo la procedura Impostazione delle autorizzazioni per l'accesso ai siti Web nella Guida per l'utente di Amazon Simple Storage Service.

Creazione di un pool di identità

Nella console Amazon Cognito, crea un pool di identità Amazon Cognito, come descritto the section called “Fase 1: creare un pool di identità di Amazon Cognito” nell'argomento Guida introduttiva allo script del browser.

Durante la creazione del pool di identità, prendi nota del nome del pool di identità e del nome del ruolo dell'identità non autenticata.

Configurare autorizzazioni ruolo

Per consentire la visualizzazione di album e foto, devi aggiungere le autorizzazioni a un ruolo IAM del pool di identità che hai appena creato. Iniziare con la creazione di una policy come segue.

1. Apri la console IAM.
2. Nel riquadro di navigazione sulla sinistra, selezionare Policies (Policy) e fare clic sul pulsante Create Policy (Crea policy).

3. Nella scheda JSON, immettere la seguente definizione JSON, ma sostituire BUCKET_NAME con il nome del bucket.

   JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME"
         ]
      }
   ]
}
```

4. Scegliere il pulsante Review policy (Esamina policy), assegnare un nome alla policy e fornire una descrizione (se si desidera), quindi scegliere il pulsante Create policy (Crea policy).

   Assicurati di annotare il nome in modo da poterlo trovare e associarlo al ruolo IAM in un secondo momento.

Dopo aver creato la policy, torna alla console IAM. Trova il ruolo IAM per l'identità non autenticata creata da Amazon Cognito nella precedente attività prerequisita, Creare un pool di identità. È possibile utilizzare la policy creata per aggiungere le autorizzazioni a questa identità.

Sebbene il flusso di lavoro per questa attività sia generalmente lo stesso di the section called “Fase 2: aggiungere una policy al ruolo IAM creato” dell’argomento Nozione di base su uno script di browser, vi sono alcune differenze da notare:

- Utilizza la nuova politica che hai appena creato, non una politica per Amazon Polly.
- Nella pagina Attach Permissions (Allega autorizzazioni), per individuare rapidamente la nuova policy, aprire l’elenco Filter policies (Filtra policy) e scegliere Customer managed (Gestito dal cliente).

Per ulteriori informazioni sulla creazione di un ruolo IAM, consulta Creazione di un ruolo per delegare le autorizzazioni a un AWS servizio nella Guida per l'utente IAM.

Configurazione CORS

Prima che lo script del browser possa accedere al bucket Amazon S3, devi configurarne la configurazione CORS come segue.

> **Important**
>
> Nella nuova console S3, la configurazione CORS deve essere JSON.

JSON

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

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>HEAD</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

Crea album e Carica foto

Poiché questo esempio consente solo agli utenti di visualizzare le foto già presenti nel bucket, è necessario creare alcuni album nel bucket e caricarvi le foto.

> **Note**
>
> Per questo esempio, i nomi dei file dei file di foto devono iniziare con una singola sottolineatura ("_"). Questo carattere è importante più tardi per i filtri. Inoltre, assicurarsi di rispettare il copyright dei proprietari delle foto.

1. Nella console Amazon S3, apri il bucket creato in precedenza.
2. Nella scheda Overview (Panoramica) scegliere il pulsante Create folder (Crea cartella) per creare le cartelle. Per questo esempio, assegnare alle cartelle i nomi "album1", "album2" e "album3".
3. Per album1 e quindi album2, selezionare la cartella e quindi caricare le foto come segue:
   a. Scegliere il pulsante Upload (Carica).
   b. Trascinare o scegliere i file di foto che si desidera utilizzare, quindi scegliere Next (Avanti).
   c. Nella sezione Manage public permissions (Gestisci autorizzazioni pubbliche), scegliere Grant public read access to this object(s) (Concedi accesso in lettura a questo oggetto/i).
   d. Scegliere il pulsante Upload (Caricamento) (nell'angolo in basso a sinistra).
4. Lasciare album3 vuoto.

Definizione della pagina Web

L'HTML per l'applicazione di visualizzazione delle foto consiste di un elemento `<div>` in cui lo script del browser crea l'interfaccia di visualizzazione. Il primo elemento `<script>` aggiunge l'SDK allo script del browser. Il secondo `<script>` elemento aggiunge il JavaScript file esterno che contiene il codice di script del browser.

Per questo esempio, il file è denominato `PhotoViewer.js` e si trova nella stessa cartella del file HTML. Per trovare l'SDK_VERSION_NUMBER corrente, consulta l'API Reference for the SDK consultabile nella API Reference Guide. JavaScript AWS SDK per 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>
```

Configurazione dell'SDK

Ottenere le credenziali necessarie per configurare il kit SDK chiamando il metodo `CognitoIdentityCredentials`. Devi fornire l'ID del pool di identità di Amazon Cognito. Successivamente, crea un oggetto di servizio `AWS.S3`.

```
// **DO THIS**:
//   Replace BUCKET_NAME with the bucket name.
//
var albumBucketName = "BUCKET_NAME";

// **DO THIS**:
//   Replace this block of code with the sample code located at:
//   Cognito -- Manage Identity Pools -- [identity_pool_name] -- Sample Code --
 JavaScript
//
// Initialize the Amazon Cognito credentials provider
AWS.config.region = "REGION"; // Region
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

// Create a new service object
var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

// A utility function to create HTML.
function getHtml(template) {
```

```
  return template.join("\n");
}
```

Il resto del codice in questo esempio definisce le seguenti funzioni per raccogliere e presentare le informazioni relative ad album e foto nel bucket.

- `listAlbums`
- `viewAlbum`

Elenco di album nel bucket.

Per elencare gli album esistenti nel bucket, la funzione `listAlbums` dell'applicazione richiama il metodo `listObjects` dell'oggetto di servizio `AWS.S3`. La funzione utilizza la proprietà `CommonPrefixes` in modo che la chiamata restituirà solo gli oggetti utilizzati come album (ovvero le cartelle).

Il resto della funzione prende l'elenco degli album dal bucket Amazon S3 e genera il codice HTML necessario per visualizzare l'elenco degli album sulla pagina Web.

```
// List the photo albums that exist in the bucket.
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          '<button style="margin:5px;" onclick="viewAlbum(\'' +
            albumName +
            "')\">",
          albumName,
          "</button>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml(["<p>Click on an album name to view it.</p>"])
        : "<p>You do not have any albums. Please Create album.";
```

```
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
      ];
      document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    }
  });
}
```

Visualizzazione di un album

Per visualizzare il contenuto di un album nel bucket Amazon S3, la `viewAlbum` funzione dell'applicazione prende il nome di un album e crea la chiave Amazon S3 per quell'album. La funzione chiama quindi il metodo `listObjects` dell'oggetto di servizio `AWS.S3` per ottenere un elenco di tutti gli oggetti (foto) nell'album.

Il resto della funzione acquisisce l'elenco degli oggetti dall'album e genera il codice HTML necessario per visualizzare le foto nella pagina Web.

```
// Show the photos that exist in an album.
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Request instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        "<br/>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
```

```
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>The following photos are present.</p>"
      : "<p>There are no photos in this album.</p>";
    var htmlTemplate = [
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      "<h2>",
      "End of Album: " + albumName,
      "</h2>",
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
    ];
    document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    document
      .getElementsByTagName("img")[0]
      .setAttribute("style", "display:none;");
  });
}
```

Visualizzazione di foto in un bucket Amazon S3: codice completo

Questa sezione contiene l'HTML completo e il JavaScript codice per l'esempio in cui è possibile visualizzare le foto in un bucket Amazon S3. Consulta la sezione padre per i dettagli e i prerequisiti.

Il codice HTML per l'esempio:

```
<!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>
```

Questo codice di esempio può essere trovato qui su. GitHub

Il codice dello script del browser come esempio:

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

Questo codice di esempio può essere trovato qui su GitHub.

Caricamento di foto su Amazon S3 da un browser

Questo esempio di codice dello script di browser illustra:

- Come creare un'applicazione browser che consenta agli utenti di creare album fotografici in un bucket Amazon S3 e caricare foto negli album.

Lo scenario

In questo esempio, una semplice pagina HTML fornisce un'applicazione basata su browser per la creazione di album fotografici in un bucket Amazon S3 in cui caricare foto. L'applicazione consente di eliminare le foto e gli album aggiunti.

Lo script del browser utilizza l'SDK JavaScript per interagire con un bucket Amazon S3. Utilizza i seguenti metodi della classe client Amazon S3 per abilitare l'applicazione per album fotografici:

- `listObjects`
- `headObject`
- `putObject`
- `upload`
- `deleteObject`
- `deleteObjects`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Nella console Amazon S3, crea un bucket Amazon S3 che utilizzerai per archiviare le foto nell'album. Per ulteriori informazioni sulla creazione di un bucket nella console, consulta Creating a Bucket nella Amazon Simple Storage Service User Guide. Verifica di disporre di entrambe le autorizzazioni Read (Leggi) e Write (Scrivi) su Objects (Oggetti). Per ulteriori informazioni sull'impostazione delle autorizzazioni per i bucket, consulta Impostazione delle autorizzazioni per l'accesso al sito Web.
- Nella console Amazon Cognito, crea un pool di identità Amazon Cognito utilizzando Federated Identities con accesso abilitato per gli utenti non autenticati nella stessa regione del bucket Amazon S3. È necessario includere l'ID del pool di identità nel codice per ottenere le credenziali per lo script di browser. Per ulteriori informazioni sulle identità federate di Amazon Cognito, consulta Amazon Cognito Identity Pools (Federated Identites) nella Amazon Cognito Developer Guide.
- Nella console IAM, trova il ruolo IAM creato da Amazon Cognito per gli utenti non autenticati. Aggiungi la seguente policy per concedere autorizzazioni di lettura e scrittura a un bucket Amazon

S3. Per ulteriori informazioni sulla creazione di un ruolo IAM, consulta Creating a Role to Delegate Permissions to an AWS Service nella IAM User Guide.

Utilizza questa politica di ruolo per il ruolo IAM creato da Amazon Cognito per utenti non autenticati.

> **Warning**
>
> Se si abilita l'accesso per gli utenti non autenticati, sarà possibile concedere l'accesso in scrittura al bucket, e a tutti gli oggetti nel bucket, a chiunque nel mondo. Questa sicurezza è utile in questo esempio per rimanere focalizzati sugli obiettivi principali dell'esempio. In molte situazioni in tempo reale, tuttavia, aumentare i livelli di sicurezza, ad esempio utilizzando gli utenti autenticati e proprietà dell'oggetto, è altamente consigliabile.

JSON

```
{
   "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/*"
         ]
      }
   ]
}
```

Configurazione di CORS

Prima che lo script del browser possa accedere al bucket Amazon S3, devi prima configurarne la configurazione CORS come segue.

> **Important**
>
> Nella nuova console S3, la configurazione CORS deve essere JSON.

JSON

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

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>POST</AllowedMethod>
        <AllowedMethod>GET</AllowedMethod>
```

```
        <AllowedMethod>PUT</AllowedMethod>
        <AllowedMethod>DELETE</AllowedMethod>
        <AllowedMethod>HEAD</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
        <ExposeHeader>ETag</ExposeHeader>
    </CORSRule>
</CORSConfiguration>
```

La pagina Web

L'HTML per l'applicazione del caricamento delle foto è costituito da un elemento ** ** all'interno dello script di browser che crea l'interfaccia utente per il caricamento. Il primo elemento **<script>** aggiunge l'SDK allo script di browser. Il secondo **<script>**elemento aggiunge il JavaScript file esterno che contiene il codice dello script del browser.

```
<!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>
```

Configurazione dell'SDK

Ottieni le credenziali necessarie per configurare l'SDK chiamando il `CognitoIdentityCredentials` metodo, fornendo l'ID del pool di identità di Amazon Cognito. Successivamente, crea un oggetto di servizio`AWS.S3`.

```
var albumBucketName = "BUCKET_NAME";
var bucketRegion = "REGION";
var IdentityPoolId = "IDENTITY_POOL_ID";

AWS.config.update({
  region: bucketRegion,
  credentials: new AWS.CognitoIdentityCredentials({
    IdentityPoolId: IdentityPoolId,
  }),
});

var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});
```

Quasi tutto il resto del codice in questo esempio è organizzato in una serie di funzioni in grado di raccogliere e presentare le informazioni relative agli album nel bucket, caricare e visualizzare le foto caricate negli album ed eliminare foto e album. Queste funzioni sono:

- `listAlbums`
- `createAlbum`
- `viewAlbum`
- `addPhoto`
- `deleteAlbum`
- `deletePhoto`

Elenco di album nel bucket.

L'applicazione crea album nel bucket Amazon S3 come oggetti le cui chiavi iniziano con una barra, che indica che l'oggetto funziona come una cartella. Per elencare tutti gli album esistenti nel bucket, la funzione dell'applicazione `listAlbums` richiama il metodo `listObjects` dell'oggetto di servizio `AWS.S3` durante l'utilizzo di `commonPrefix` in modo che la chiamata restituirà solo gli oggetti utilizzati come album.

Il resto della funzione prende l'elenco degli album dal bucket Amazon S3 e genera il codice HTML necessario per visualizzare l'elenco degli album nella pagina Web. Consente inoltre l'eliminazione e l'apertura di singoli album.

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

Creazione di un album nel bucket

Per creare un album nel bucket Amazon S3, la `createAlbum` funzione dell'applicazione convalida innanzitutto il nome dato al nuovo album per assicurarsi che contenga caratteri adatti. La funzione forma quindi una chiave oggetto Amazon S3, passandola al `headObject` metodo dell'oggetto

di servizio Amazon S3. Questo metodo restituisce i metadati per la chiave specificata, perciò se restituisce i dati, allora un oggetto con quella chiave già esiste.

Se l'album non esiste, la funzione richiama il metodo `putObject` dell'oggetto di servizio `AWS.S3` per creare l'album. È quindi richiama la funzione `viewAlbum` per visualizzare il nuovo album vuoto.

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

Visualizzazione di un album

Per visualizzare il contenuto di un album nel bucket Amazon S3, la `viewAlbum` funzione dell'applicazione prende il nome di un album e crea la chiave Amazon S3 per quell'album. La funzione chiama quindi il metodo `listObjects` dell'oggetto di servizio `AWS.S3` per ottenere un elenco di tutti gli oggetti (foto) nell'album.

Il resto della funzione acquisisce l'elenco degli oggetti (foto) dall'album e genera il codice HTML necessario per visualizzare le foto nella pagina Web. Consente inoltre l'eliminazione di singole foto e la navigazione nell'elenco dell'album.

```
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Response instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span onclick=\"deletePhoto('" +
          albumName +
          "','" +
          photoKey +
          "')\">",
        "X",
        "</span>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>Click on the X to delete the photo</p>"
      : "<p>You do not have any photos in this album. Please add photos.</p>";
    var htmlTemplate = [
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
```

```
      "</div>",
      '<input id="photoupload" type="file" accept="image/*">',
      '<button id="addphoto" onclick="addPhoto(\'' + albumName + "')\">",
      "Add Photo",
      "</button>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
    ];
    document.getElementById("app").innerHTML = getHtml(htmlTemplate);
  });
}
```

Aggiunta di foto a un album

Per caricare una foto in un album nel bucket Amazon S3, la `addPhoto` funzione dell'applicazione utilizza un elemento di selezione dei file nella pagina Web per identificare un file da caricare. È quindi forma una chiave per le foto da caricare dall'attuale nome dell'album e nome del file.

La funzione richiama il `upload` metodo dell'oggetto del servizio Amazon S3 per caricare la foto. Dopo aver caricato la foto, la funzione visualizza nuovamente l'album in modo che vengano visualizzate le foto caricate.

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

Eliminazione di una foto

Per eliminare una foto da un album nel bucket Amazon S3, la `deletePhoto` funzione dell'applicazione richiama il `deleteObject` metodo dell'oggetto di servizio Amazon S3. Ciò elimina la foto specificata dal valore `photoKey` passato alla funzione.

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

Eliminazione di un album

Per eliminare un album nel bucket Amazon S3, la `deleteAlbum` funzione dell'applicazione richiama il `deleteObjects` metodo dell'oggetto di servizio Amazon S3.

```
function deleteAlbum(albumName) {
  var albumKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your album: ", err.message);
    }
    var objects = data.Contents.map(function (object) {
      return { Key: object.Key };
    });
```

```
    s3.deleteObjects(
      {
        Delete: { Objects: objects, Quiet: true },
      },
      function (err, data) {
        if (err) {
          return alert("There was an error deleting your album: ", err.message);
        }
        alert("Successfully deleted album.");
        listAlbums();
      }
    );
  });
}
```

Caricamento di foto su Amazon S3: codice completo

Questa sezione contiene l'HTML completo e il JavaScript codice per l'esempio in cui le foto vengono caricate su un album di foto Amazon S3. Consulta la sezione padre per i dettagli e i prerequisiti.

Il codice HTML per l'esempio:

```
<!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>
```

Questo codice di esempio è disponibile qui su GitHub.

Il codice dello script del browser come esempio:

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

Questo codice di esempio può essere trovato qui su GitHub.

Esempi di Amazon S3 Node.js

I seguenti argomenti mostrano esempi di come AWS SDK per JavaScript può essere usato per interagire con i bucket Amazon S3 utilizzando Node.js.

Argomenti

- Creazione e utilizzo dei bucket di Amazon S3
- Configurazione dei bucket di Amazon S3
- Gestione delle autorizzazioni di accesso dei bucket di Amazon S3
- Utilizzo delle policy per i bucket di Amazon S3

- Utilizzo di un bucket di Amazon S3 come host Web statico

Creazione e utilizzo dei bucket di Amazon S3

Questo esempio di codice di Node.js illustra:

- Come ottenere e visualizzare un elenco di bucket Amazon S3 nel tuo account.
- Come creare un bucket Amazon S3.
- Come caricare un oggetto in un bucket specifico.

Lo scenario

In questo esempio, una serie di moduli Node.js vengono utilizzati per ottenere un elenco di bucket Amazon S3 esistenti, creare un bucket e caricare un file in un bucket specificato. Questi moduli Node.js utilizzano l'SDK per JavaScript ottenere informazioni e caricare file su un bucket Amazon S3 utilizzando questi metodi della classe client Amazon S3:

- `listBuckets`
- `createBucket`
- `listObjects`
- `upload`
- `deleteBucket`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Configurazione dell'SDK

Configura l'SDK per JavaScript creando un oggetto di configurazione globale, quindi impostando la regione per il codice. Nell'esempio, la regione è impostata su `us-west-2`.

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

Visualizzazione di un elenco di bucket Amazon S3

Crea un modulo Node.js con il nome del file `s3_listbuckets.js`. Assicurati di configurare l'SDK come mostrato in precedenza. Per accedere ad Amazon Simple Storage Service, crea un oggetto `AWS.S3` servizio. Chiama il `listBuckets` metodo dell'oggetto di servizio Amazon S3 per recuperare un elenco dei tuoi bucket. Il parametro `data` della funzione di callback dispone di una proprietà `Buckets` che contiene una vasta gamma di mappe per rappresentare i bucket. Visualizza l'elenco dei bucket eseguendo l'accesso alla console.

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

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Call S3 to list the buckets
s3.listBuckets(function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Buckets);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_listbuckets.js
```

Questo codice di esempio può essere trovato qui. GitHub

Creazione di un bucket Amazon S3

Crea un modulo Node.js con il nome del file `s3_createbucket.js`. Assicurati di configurare l'SDK come mostrato in precedenza. Crea un oggetto di servizio`AWS.S3`. Il modulo richiede un singolo argomento della riga di comando per specificare un nome per il nuovo bucket.

Aggiungi una variabile per contenere i parametri utilizzati per chiamare il `createBucket` metodo dell'oggetto di servizio Amazon S3, incluso il nome del bucket appena creato. La funzione di callback registra la posizione del nuovo bucket nella console dopo che Amazon S3 lo ha creato correttamente.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_createbucket.js BUCKET_NAME
```

Questo codice di esempio può essere trovato qui su. GitHub

Caricamento di un file in un bucket Amazon S3

Crea un modulo Node.js con il nome del file `s3_upload.js`. Assicurati di configurare l'SDK come mostrato in precedenza. Crea un oggetto di servizio`AWS.S3`. Il modulo richiederà due argomenti della

riga di comando, il primo per specificare il bucket di destinazione e il secondo per specificare il file da caricare.

Crea una variabile con i parametri necessari per chiamare il `upload` metodo dell'oggetto del servizio Amazon S3. Fornisci il nome del bucket di destinazione nel parametro `Bucket`. Il parametro `Key` è impostato sul nome del file selezionato, che è possibile ottenere utilizzando il modulo di Node.js. `path`. Il parametro Body è impostato sui contenuti del file, che è possibile ottenere utilizzando `createReadStream` dal modulo di Node.js. `fs`.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_upload.js BUCKET_NAME FILE_NAME
```

Questo codice di esempio può essere trovato qui su GitHub.

Elencare oggetti in un bucket Amazon S3

Crea un modulo Node.js con il nome del file `s3_listobjects.js`. Assicurati di configurare l'SDK come mostrato in precedenza. Crea un oggetto di servizio`AWS.S3`.

Aggiungi una variabile per contenere i parametri utilizzati per chiamare il `listObjects` metodo dell'oggetto del servizio Amazon S3, incluso il nome del bucket da leggere. La funzione di callback registra un elenco di oggetti (file) o un messaggio di errore.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_listobjects.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Eliminazione di un bucket Amazon S3

Crea un modulo Node.js con il nome del file `s3_deletebucket.js`. Assicurati di configurare l'SDK come mostrato in precedenza. Crea un oggetto di servizio`AWS.S3`.

Aggiungi una variabile per contenere i parametri utilizzati per chiamare il `createBucket` metodo dell'oggetto del servizio Amazon S3, incluso il nome del bucket da eliminare. Il bucket deve essere vuoto per essere eliminato. La funzione di callback registra un messaggio di successo o di errore.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_deletebucket.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Configurazione dei bucket di Amazon S3

Questo esempio di codice di Node.js illustra:

- Come configurare le autorizzazioni della condivisione delle risorse multiorigine (CORS) per un bucket.

Lo scenario

In questo esempio, una serie di moduli di Node.js vengono utilizzati per elencare i bucket Amazon S3 e configurare CORS e la registrazione dei bucket. I moduli Node.js utilizzano l'SDK JavaScript per configurare un bucket Amazon S3 selezionato utilizzando questi metodi della classe client Amazon S3:

- `getBucketCors`
- `putBucketCors`

Per ulteriori informazioni sull'utilizzo della configurazione CORS con un bucket Amazon S3, consulta Cross-Origin Resource Sharing (CORS) nella Guida per l'utente di Amazon Simple Storage Service.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Configurazione dell'SDK

Configura l'SDK per JavaScript creando un oggetto di configurazione globale, quindi impostando la regione per il codice. Nell'esempio, la regione è impostata su `us-west-2`.

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

Recuperare una configurazione CORS del bucket

Crea un modulo Node.js con il nome del file `s3_getcors.js`. Il modulo richiede un singolo argomento della riga di comando per specificare il bucket di cui desideri la configurazione CORS. Assicurati di configurare l'SDK come mostrato in precedenza. Crea un oggetto di servizio`AWS.S3`.

L'unico parametro che deve essere passato è il nome del bucket selezionato quando si chiama il metodo `getBucketCors`. Se il bucket ha attualmente una configurazione CORS, tale configurazione viene restituita da Amazon S3 come proprietà `CORSRules` del parametro passato `data` alla funzione di callback.

Se il bucket selezionato non dispone di alcuna configurazione CORS, tali informazioni vengono restituite alla funzione di callback nel parametro `error`.

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

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Set the parameters for S3.getBucketCors
var bucketParams = { Bucket: process.argv[2] };

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_getcors.js BUCKET_NAME
```

Questo codice di esempio può essere trovato qui su. GitHub

Impostazione di una configurazione CORS del bucket

Crea un modulo Node.js con il nome del file `s3_setcors.js`. Il modulo richiede più argomenti della riga di comando, il primo dei quali specifica il bucket di cui desideri impostare la configurazione CORS. Altri argomenti enumerano i metodi HTTP (POST, GET, PUT, PATCH, DELETE, POST) che desideri consentire per il bucket. Configura l'SDK come mostrato in precedenza.

Crea un oggetto di servizio`AWS.S3`. Successivamente crea un oggetto JSON per conservare i valori della configurazione CORS, secondo quanto richiesto dal metodo `putBucketCors` dell'oggetto di servizio `AWS.S3`. Specifica `"Authorization"` per il valore `AllowedHeaders` e `"*"` per il valore `AllowedOrigins`. Inizialmente imposta il valore di `AllowedMethods` come matrice vuota.

Specifica i metodi consentiti come parametri della riga di comando per il modulo di Node.js, aggiungendo ciascuno dei metodi che corrispondono a uno dei parametri. Aggiungi la configurazione CORS risultante alla gamma di configurazioni contenute nel parametro `CORSRules`. Specifica il bucket da configurare per CORS nel parametro `Bucket`.

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

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create initial parameters JSON for putBucketCors
var thisConfig = {
  AllowedHeaders: ["Authorization"],
  AllowedMethods: [],
  AllowedOrigins: ["*"],
  ExposeHeaders: [],
  MaxAgeSeconds: 3000,
};

// Assemble the list of allowed methods based on command line parameters
var allowedMethods = [];
process.argv.forEach(function (val, index, array) {
  if (val.toUpperCase() === "POST") {
```

```
    allowedMethods.push("POST");
  }
  if (val.toUpperCase() === "GET") {
    allowedMethods.push("GET");
  }
  if (val.toUpperCase() === "PUT") {
    allowedMethods.push("PUT");
  }
  if (val.toUpperCase() === "PATCH") {
    allowedMethods.push("PATCH");
  }
  if (val.toUpperCase() === "DELETE") {
    allowedMethods.push("DELETE");
  }
  if (val.toUpperCase() === "HEAD") {
    allowedMethods.push("HEAD");
  }
});

// Copy the array of allowed methods into the config object
thisConfig.AllowedMethods = allowedMethods;
// Create array of configs then add the config object to it
var corsRules = new Array(thisConfig);

// Create CORS params
var corsParams = {
  Bucket: process.argv[2],
  CORSConfiguration: { CORSRules: corsRules },
};

// set the new CORS configuration on the selected bucket
s3.putBucketCors(corsParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    // update the displayed CORS config for the selected bucket
    console.log("Success", data);
  }
});
```

Per eseguire l'esempio, digita la seguente stringa nella riga di comando, tra cui uno o più metodi HTTP come mostrato.

```
node s3_setcors.js BUCKET_NAME get put
```

Questo codice di esempio può essere trovato qui su GitHub.

Gestione delle autorizzazioni di accesso dei bucket di Amazon S3

Questo esempio di codice di Node.js illustra:

- Come recuperare o impostare l'elenco di controllo degli accessi per un bucket Amazon S3.

Lo scenario

In questo esempio, un modulo di Node.js viene utilizzato per visualizzare la lista di controllo accessi (ACL) del bucket per un bucket selezionato e applicare le modifiche alla lista per un bucket selezionato. Il modulo Node.js utilizza l'SDK per JavaScript gestire le autorizzazioni di accesso ai bucket Amazon S3 utilizzando questi metodi della classe client Amazon S3:

- `getBucketAcl`
- `putBucketAcl`

Per ulteriori informazioni sugli elenchi di controllo degli accessi per i bucket Amazon S3, consulta Managing Access with nella Amazon ACLs Simple Storage Service User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Configurazione dell'SDK

Configura l'SDK per JavaScript creando un oggetto di configurazione globale, quindi impostando la regione per il codice. Nell'esempio, la regione è impostata su `us-west-2`.

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

Recupero della lista di controllo accessi attuale del bucket

Crea un modulo Node.js con il nome del file `s3_getbucketacl.js`. Il modulo richiede un singolo argomento della riga di comando per specificare il bucket di cui desideri la configurazione ACL. Assicurati di configurare l'SDK come mostrato in precedenza.

Crea un oggetto di servizio`AWS.S3`. L'unico parametro che deve essere passato è il nome del bucket selezionato quando si chiama il metodo `getBucketAcl`. La configurazione corrente della lista di controllo degli accessi viene restituita da Amazon S3 nel `data` parametro passato alla funzione di callback.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_getbucketacl.js BUCKET_NAME
```

Questo codice di esempio può essere trovato qui su. GitHub

Utilizzo delle policy per i bucket di Amazon S3

Questo esempio di codice di Node.js illustra:

- Come recuperare la policy sui bucket di un bucket Amazon S3.
- Come aggiungere o aggiornare la policy sui bucket di un bucket Amazon S3.
- Come eliminare la policy sui bucket di un bucket Amazon S3.

Lo scenario

In questo esempio, una serie di moduli Node.js vengono utilizzati per recuperare, impostare o eliminare una policy bucket su un bucket Amazon S3. I moduli Node.js utilizzano l'SDK per JavaScript configurare la policy per un bucket Amazon S3 selezionato utilizzando questi metodi della classe client Amazon S3:

- `getBucketPolicy`
- `putBucketPolicy`
- `deleteBucketPolicy`

Per ulteriori informazioni sulle policy dei bucket per i bucket Amazon S3, consulta Using Bucket Policies and User Policies nella Amazon Simple Storage Service User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Configurazione dell'SDK

Configura l'SDK per JavaScript creando un oggetto di configurazione globale, quindi impostando la regione per il codice. Nell'esempio, la regione è impostata su `us-west-2`.

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

Recupero della policy dei bucket attuale

Crea un modulo Node.js con il nome del file `s3_getbucketpolicy.js`. Il modulo richiede un singolo argomento della riga di comando per specificare il bucket di cui desideri la policy. Assicurati di configurare l'SDK come mostrato in precedenza.

Crea un oggetto di servizio`AWS.S3`. L'unico parametro che deve essere passato è il nome del bucket selezionato quando si chiama il metodo `getBucketPolicy`. Se il bucket ha attualmente una policy, tale policy viene restituita da Amazon S3 nel parametro passato alla `data` funzione di callback.

Se il bucket selezionato non dispone di alcuna policy, tali informazioni vengono restituite alla funzione di callback nel parametro `error`.

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

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to retrieve policy for selected bucket
s3.getBucketPolicy(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
```

```
    console.log("Success", data.Policy);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_getbucketpolicy.js BUCKET_NAME
```

Questo codice di esempio può essere trovato qui su. GitHub

Impostazione di una policy dei bucket semplice

Crea un modulo Node.js con il nome del file `s3_setbucketpolicy.js`. Il modulo richiede un singolo argomento della riga di comando per specificare il bucket di cui desideri applicare la policy. Configura l'SDK come mostrato in precedenza.

Crea un oggetto di servizio`AWS.S3`. Le policy dei bucket sono specificate in un oggetto JSON. In primo luogo, crea un oggetto JSON che contenga tutti i valori per specificare la policy tranne il valore `Resource` che identifica il bucket.

Formatta la stringa `Resource` necessaria per la policy, incorporando il nome del bucket selezionato. Inserisci la stringa nell'oggetto JSON. Prepara i parametri per il metodo `putBucketPolicy`, tra cui il nome del bucket e la policy JSON convertiti a un valore di stringa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_setbucketpolicy.js BUCKET_NAME
```

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione di una policy dei bucket

Crea un modulo Node.js con il nome del file `s3_deletebucketpolicy.js`. Il modulo richiede un singolo argomento della riga di comando per specificare il bucket di cui desideri eliminare la policy. Configura l'SDK come mostrato in precedenza.

Crea un oggetto di servizio`AWS.S3`. L'unico parametro che deve essere passato quando si chiama il metodo `deleteBucketPolicy` è il nome del bucket selezionato.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_deletebucketpolicy.js BUCKET_NAME
```

Questo codice di esempio può essere trovato qui su GitHub.

Utilizzo di un bucket di Amazon S3 come host Web statico

Questo esempio di codice di Node.js illustra:

- Come configurare un bucket Amazon S3 come host web statico.

Lo scenario

In questo esempio, una serie di moduli di Node.js viene utilizzata per configurare tutti i bucket per espletare funzioni di host Web statico. I moduli Node.js utilizzano l'SDK JavaScript per configurare un bucket Amazon S3 selezionato utilizzando questi metodi della classe client Amazon S3:

- `getBucketWebsite`
- `putBucketWebsite`
- `deleteBucketWebsite`

Per ulteriori informazioni sull'utilizzo di un bucket Amazon S3 come host Web statico, consulta Hosting a Static Website on Amazon S3 nella Amazon Simple Storage Service User Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Configurazione dell'SDK

Configura l'SDK per JavaScript creando un oggetto di configurazione globale, quindi impostando la regione per il codice. Nell'esempio, la regione è impostata su `us-west-2`.

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

Recupero dell'attuale configurazione del sito Web di un bucket

Crea un modulo Node.js con il nome del file `s3_getbucketwebsite.js`. Il modulo richiede un singolo argomento della riga di comando per specificare il bucket di cui desideri la configurazione del sito Web. Configura l'SDK come mostrato in precedenza.

Crea un oggetto di servizio`AWS.S3`. Crea una funzione che recupera l'attuale configurazione del sito Web di un bucket per il bucket selezionato nell'elenco dei bucket. L'unico parametro che deve essere passato è il nome del bucket selezionato quando si chiama il metodo `getBucketWebsite`. Se il bucket ha attualmente una configurazione del sito Web, tale configurazione viene restituita da Amazon S3 nel `data` parametro passato alla funzione di callback.

Se il bucket selezionato non dispone di alcuna configurazione del sito Web, tali informazioni vengono restituite alla funzione di callback nel parametro `err`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };

// call S3 to retrieve the website configuration for selected bucket
s3.getBucketWebsite(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_getbucketwebsite.js BUCKET_NAME
```

Questo codice di esempio può essere trovato qui su. GitHub

Impostazione della configurazione del sito Web di un bucket

Crea un modulo Node.js con il nome del file `s3_setbucketwebsite.js`. Assicurati di configurare l'SDK come mostrato in precedenza. Crea un oggetto di servizio`AWS.S3`.

Crea una funzione che applica una configurazione del sito Web di un bucket. La configurazione consente al bucket selezionato di fungere da host Web statico. Le configurazioni dei siti Web sono specificate in formato JSON. In primo luogo, crea un oggetto JSON che contenga tutti i valori per specificare la configurazione del sito Web, salvo per il valore `Key` che identifica il documento di errore e il valore `Suffix` che identifica il documento di indice.

Inserisci i valori degli elementi di input di testo nell'oggetto JSON. Prepara i parametri per il metodo `putBucketWebsite`, tra cui il nome del bucket e la configurazione del sito Web JSON.

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

// Create S3 service object
```

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

// Create JSON for putBucketWebsite parameters
var staticHostParams = {
  Bucket: "",
  WebsiteConfiguration: {
    ErrorDocument: {
      Key: "",
    },
    IndexDocument: {
      Suffix: "",
    },
  },
};

// Insert specified bucket name and index and error documents into params JSON
// from command line arguments
staticHostParams.Bucket = process.argv[2];
staticHostParams.WebsiteConfiguration.IndexDocument.Suffix = process.argv[3];
staticHostParams.WebsiteConfiguration.ErrorDocument.Key = process.argv[4];

// set the new website configuration on the selected bucket
s3.putBucketWebsite(staticHostParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    // update the displayed website configuration for the selected bucket
    console.log("Success", data);
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_setbucketwebsite.js BUCKET_NAME INDEX_PAGE ERROR_PAGE
```

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione della configurazione del sito Web di un bucket

Crea un modulo Node.js con il nome del file `s3_deletebucketwebsite.js`. Assicurati di configurare l'SDK come mostrato in precedenza. Crea un oggetto di servizio`AWS.S3`.

Crea una funzione che elimina la configurazione del sito Web per il bucket selezionato. L'unico parametro che deve essere passato quando si chiama il metodo `deleteBucketWebsite` è il nome del bucket selezionato.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node s3_deletebucketwebsite.js BUCKET_NAME
```

Questo codice di esempio può essere trovato qui su GitHub.

Esempi di servizi Amazon Simple Email

Amazon Simple Email Service (Amazon SES) Simple Email Service (Amazon SES) è un servizio di invio e-mail basato sul cloud progettato per aiutare gli esperti di marketing digitale e gli sviluppatori di applicazioni a inviare e-mail di marketing, notifiche e transazionali. Si tratta di un servizio affidabile, a costo ridotto per aziende di tutte le dimensioni che utilizzano la posta elettronica per mantenere il contatto con i clienti.

L' JavaScript API per Amazon SES è esposta tramite la classe `AWS.SES` client. Per ulteriori informazioni sull'uso della classe client Amazon SES, `Class: AWS.SES`consulta il riferimento all'API.

Argomenti

- Gestione delle identità Amazon SES
- Utilizzo dei modelli di e-mail in Amazon SES
- Invio di e-mail tramite Amazon SES
- Utilizzo dei filtri degli indirizzi IP per la ricezione di e-mail in Amazon SES
- Utilizzo delle regole di ricezione in Amazon SES

Gestione delle identità Amazon SES

Questo esempio di codice di Node.js illustra:

- Come verificare gli indirizzi e-mail e i domini utilizzati con Amazon SES.
- Come assegnare la policy IAM alle tue identità Amazon SES.
- Come elencare tutte le identità Amazon SES per il tuo AWS account.
- Come eliminare le identità utilizzate con Amazon SES.

Un'identità Amazon SES è un indirizzo e-mail o un dominio che Amazon SES utilizza per inviare e-mail. Amazon SES richiede che verifichi le tue identità e-mail, confermando che le possiedi e impedendo ad altri di utilizzarle.

Per dettagli su come verificare indirizzi e-mail e domini in Amazon SES, consulta Verifying Email Addresses and Domains in Amazon SES nella Amazon Simple Email Service Developer Guide. Per informazioni sull'autorizzazione all'invio in Amazon SES, consulta Panoramica dell'autorizzazione all'invio di Amazon SES.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per verificare e gestire le identità di Amazon SES. I moduli Node.js utilizzano l'SDK per JavaScript verificare indirizzi e-mail e domini, utilizzando questi metodi della `AWS.SES` classe client:

- `listIdentities`
- `deleteIdentity`
- `verifyEmailIdentity`
- `verifyDomainIdentity`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni su come fornire file JSON di credenziali, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Configurazione dell'SDK

Configura l'SDK per JavaScript creando un oggetto di configurazione globale, quindi impostando la regione per il codice. Nell'esempio, la regione è impostata su `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
```

```
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Elenco delle tue identità

In questo esempio, utilizza un modulo Node.js per elencare gli indirizzi e-mail e i domini da utilizzare con Amazon SES. Crea un modulo Node.js con il nome del file `ses_listidentities.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire `IdentityType` e altri parametri per il metodo `listIdentities` della classe client `AWS.SES`. Per chiamare il `listIdentities` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SES, passando l'oggetto parameters.

Quindi gestisci l'oggetto `response` nel callback della promessa. L'oggetto `data` restituito dalla promessa contiene una matrice di identità dominio come specificato dal parametro `IdentityType`.

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

// Create listIdentities params
var params = {
  IdentityType: "Domain",
  MaxItems: 10,
};

// Create the promise and SES service object
var listIDsPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listIdentities(params)
  .promise();

// Handle promise's fulfilled/rejected states
listIDsPromise
  .then(function (data) {
    console.log(data.Identities);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node ses_listidentities.js
```

Questo codice di esempio può essere trovato qui. GitHub

Verifica di un'identità indirizzo e-mail

In questo esempio, utilizza un modulo Node.js per verificare i mittenti di posta elettronica da utilizzare con Amazon SES. Crea un modulo Node.js con il nome del file `ses_verifyemailidentity.js`. Configura l'SDK come mostrato in precedenza. Per accedere ad Amazon SES, crea un oggetto `AWS.SES` di servizio.

Crea un oggetto per trasferire il parametro `EmailAddress` per il metodo `verifyEmailIdentity` della classe client `AWS.SES`. Per chiamare il metodo `verifyEmailIdentity`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Il dominio viene aggiunto ad Amazon SES per essere verificato.

```
node ses_verifyemailidentity.js
```

Questo codice di esempio può essere trovato qui GitHub.

Verifica di un'identità dominio

In questo esempio, utilizza un modulo Node.js per verificare i domini e-mail da utilizzare con Amazon SES. Crea un modulo Node.js con il nome del file `ses_verifydomainidentity.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire il parametro `Domain` per il metodo `verifyDomainIdentity` della classe client `AWS.SES`. Per chiamare il `verifyDomainIdentity` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SES, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Il dominio viene aggiunto ad Amazon SES per essere verificato.

```
node ses_verifydomainidentity.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di identità

In questo esempio, utilizza un modulo Node.js per eliminare gli indirizzi e-mail o i domini utilizzati con Amazon SES. Crea un modulo Node.js con il nome del file `ses_deleteidentity.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire il parametro `Identity` per il metodo `deleteIdentity` della classe client `AWS.SES`. Per chiamare il `deleteIdentity` metodo, crea un oggetto di servizio Amazon SES `request` per richiamare un oggetto di servizio Amazon SES, passando i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node ses_deleteidentity.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Utilizzo dei modelli di e-mail in Amazon SES

Questo esempio di codice di Node.js illustra:

- Ottenere un elenco di tutti modelli di e-mail.
- Recuperare e aggiornare i modelli di e-mail.
- Creare ed eliminare i modelli di e-mail.

Amazon SES ti consente di inviare messaggi e-mail personalizzati utilizzando modelli di e-mail. Per dettagli su come creare e utilizzare modelli di e-mail in Amazon Simple Email Service, consulta Invio di e-mail personalizzate utilizzando l'API Amazon SES nella Amazon Simple Email Service Developer Guide.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per utilizzare i modelli di e-mail. I moduli Node.js utilizzano l'SDK per JavaScript creare e utilizzare modelli di e-mail utilizzando questi metodi della classe `AWS.SES` client:

- `listTemplates`
- `createTemplate`
- `getTemplate`
- `deleteTemplate`
- `updateTemplate`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla creazione di un file delle credenziali, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Elenco dei tuoi modelli di e-mail

In questo esempio, utilizza un modulo Node.js per creare un modello di e-mail da utilizzare con Amazon SES. Crea un modulo Node.js con il nome del file `ses_listtemplates.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire i parametri per il metodo `listTemplates` della classe client `AWS.SES`. Per chiamare il metodo `listTemplates`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES restituisce l'elenco dei modelli.

```
node ses_listtemplates.js
```

Questo codice di esempio può essere trovato qui GitHub.

Recupero di un modello di e-mail

In questo esempio, utilizza un modulo Node.js per ottenere un modello di e-mail da utilizzare con Amazon SES. Crea un modulo Node.js con il nome del file `ses_gettemplate.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire il parametro `TemplateName` per il metodo `getTemplate` della classe client `AWS.SES`. Per chiamare il metodo `getTemplate`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES restituisce i dettagli del modello.

```
node ses_gettemplate.js
```

Questo codice di esempio può essere trovato qui GitHub.

Creazione di un modello di e-mail

In questo esempio, utilizza un modulo Node.js per creare un modello di e-mail da utilizzare con Amazon SES. Crea un modulo Node.js con il nome del file `ses_createtemplate.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire i parametri per il metodo `createTemplate` della classe client `AWS.SES`, inclusi `TemplateName`, `HtmlPart`, `SubjectPart` e `TextPart`. Per chiamare il metodo `createTemplate`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Il modello viene aggiunto ad Amazon SES.

```
node ses_createtemplate.js
```

Questo codice di esempio può essere trovato qui GitHub.

Aggiornamento di un modello di e-mail

In questo esempio, utilizza un modulo Node.js per creare un modello di e-mail da utilizzare con Amazon SES. Crea un modulo Node.js con il nome del file `ses_updatetemplate.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire i valori dei parametri `Template` che desideri aggiornare nel modello, con il parametro `TemplateName` richiesto trasferito al metodo `updateTemplate` della classe client `AWS.SES`. Per chiamare il metodo `updateTemplate`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES restituisce i dettagli del modello.

```
node ses_updatetemplate.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di un modello di e-mail

In questo esempio, utilizza un modulo Node.js per creare un modello di e-mail da utilizzare con Amazon SES. Crea un modulo Node.js con il nome del file `ses_deletetemplate.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire il parametro `TemplateName` richiesto al metodo `deleteTemplate` della classe client `AWS.SES`. Per chiamare il metodo `deleteTemplate`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES restituisce i dettagli del modello.

```
node ses_deletetemplate.js
```

Questo codice di esempio può essere trovato qui GitHub.

Invio di e-mail tramite Amazon SES

Questo esempio di codice di Node.js illustra:

- Come inviare un'e-mail di testo o in formato HTML.
- Come inviare e-mail in base a un modello di e-mail.
- Come inviare e-mail in blocco in base a un modello di e-mail.

L'API Amazon SES offre due modi diversi per inviare un'e-mail, a seconda del livello di controllo che desideri sulla composizione del messaggio e-mail: formattato e non elaborato. Per i dettagli, consulta Invio di e-mail formattate utilizzando l'API Amazon SES e Invio di e-mail non elaborate utilizzando l'API Amazon SES.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per inviare e-mail in diversi modi. I moduli Node.js utilizzano l'SDK per JavaScript creare e utilizzare modelli di e-mail utilizzando questi metodi della classe `AWS.SES` client:

- `sendEmail`
- `sendTemplatedEmail`
- `sendBulkTemplatedEmail`

Attività prerequisite

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni su come fornire file JSON di credenziali, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Requisiti per l'invio di messaggi e-mail

Amazon SES compone un messaggio e-mail e lo mette immediatamente in coda per l'invio. Per inviare e-mail utilizzando il metodo `SES.sendEmail`, il messaggio deve soddisfare i seguenti requisiti:

- Devi inviare il messaggio da un dominio o da un indirizzo e-mail verificato. Se tenti di inviare e-mail utilizzando un dominio o un indirizzo non verificato, l'operazione genera un errore `"Email address not verified"`.
- Se il tuo account si trova ancora nella sandbox di Amazon SES, puoi inviare messaggi solo a domini o indirizzi verificati oppure a indirizzi e-mail associati al simulatore di mailbox di Amazon SES. Per ulteriori informazioni, consulta Verifying Email Addresses and Domains nella Amazon Simple Email Service Developer Guide.
- La dimensione totale del messaggio, inclusi gli allegati, deve essere inferiore a 10 MB.
- Il messaggio deve includere almeno l'indirizzo e-mail di un destinatario. L'indirizzo del destinatario può essere un indirizzo "To:" ("A:"), "CC:" ("Cc:") o "BCC:" ("Ccn:"). Se l'indirizzo e-mail di un destinatario non è valido (ovvero non è nel formato `UserName@[SubDomain.]Domain.TopLevelDomain`), l'intero messaggio viene rifiutato, anche se contiene altri destinatari validi.
- Il messaggio non può includere più di 50 destinatari, nei campi "To:" ("A:"), "CC:" ("Cc:") e "BCC:" ("Ccn:"). Se devi inviare un messaggio e-mail a un pubblico più ampio, puoi dividere l'elenco dei destinatari in gruppi di massimo 50 persone e quindi chiamare il metodo `sendEmail` più volte per inviare il messaggio a ciascun gruppo.

Invio di un'e-mail

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_sendemail.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire i valori dei parametri che definiscono l'e-mail da inviare, inclusi gli indirizzi dei mittenti e dei destinatari, l'oggetto, il corpo dell'e-mail in testo normale e in formato HTML, al metodo `sendEmail` della classe client `AWS.SES`. Per chiamare il metodo `sendEmail`, crea una

promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. L'e-mail viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendemail.js
```

Questo codice di esempio può essere trovato qui. GitHub

Invio di un'e-mail in base a un modello

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_sendtemplatedemail.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire i valori dei parametri che definiscono l'e-mail da inviare, inclusi gli indirizzi dei mittenti e dei destinatari, l'oggetto, il corpo dell'e-mail in testo normale e in formato HTML, al metodo `sendTemplatedEmail` della classe client `AWS.SES`. Per chiamare il metodo `sendTemplatedEmail`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. L'e-mail viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendtemplatedemail.js
```

Questo codice di esempio può essere trovato qui. GitHub

Invio di e-mail in blocco in base a un modello

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_sendbulktemplatedemail.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire i valori dei parametri che definiscono l'e-mail da inviare, inclusi gli indirizzi dei mittenti e dei destinatari, l'oggetto, il corpo dell'e-mail in testo normale e in formato HTML, al metodo `sendBulkTemplatedEmail` della classe client `AWS.SES`. Per chiamare il metodo `sendBulkTemplatedEmail`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. L'e-mail viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Questo codice di esempio può essere trovato qui. GitHub

Utilizzo dei filtri degli indirizzi IP per la ricezione di e-mail in Amazon SES

Questo esempio di codice di Node.js illustra:

- Come creare i filtri degli indirizzi IP per accettare o rifiutare le e-mail provenienti da un indirizzo IP o da un intervallo di indirizzi IP.
- Come elencare il tuoi i filtri degli indirizzi IP correnti.
- Come eliminare un filtro degli indirizzi IP.

In Amazon SES, un filtro è una struttura di dati composta da un nome, un intervallo di indirizzi IP e se consentire o bloccare la posta da esso. Gli indirizzi IP che desideri bloccare o consentire sono specificati come un singolo indirizzo IP o un intervallo di indirizzi IP nella notazione CIDR (Classless

Inter-Domain Routing). Per informazioni dettagliate su come Amazon SES riceve le e-mail, consulta Amazon SES Email-Receiving Concepts nella Amazon Simple Email Service Developer Guide.

Lo scenario

In questo esempio, viene utilizzata una serie di moduli Node.js per inviare e-mail in diversi modi. I moduli Node.js utilizzano l'SDK per JavaScript creare e utilizzare modelli di e-mail utilizzando questi metodi della `AWS.SES` classe client:

- `createReceiptFilter`
- `listReceiptFilters`
- `deleteReceiptFilter`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Configurazione dell'SDK

Configura l'SDK per JavaScript creando un oggetto di configurazione globale, quindi impostando la regione per il codice. Nell'esempio, la regione è impostata su `us-west-2`.

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

Creazione di un filtro degli indirizzi IP

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_createreceiptfilter.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire i valori dei parametri che definiscono il filtro IP, inclusi il nome del filtro, un indirizzo o un intervallo di indirizzi IP da filtrare e l'impostazione per consentire o bloccare il traffico e-mail proveniente dagli indirizzi filtrati. Per chiamare il metodo `createReceiptFilter`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Il filtro viene creato in Amazon SES.

```
node ses_createreceiptfilter.js
```

Questo codice di esempio può essere trovato qui GitHub.

Elenco dei tuoi filtri degli indirizzi IP

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_listreceiptfilters.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto parameters vuoto. Per chiamare il metodo `listReceiptFilters`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES restituisce l'elenco dei filtri.

```
node ses_listreceiptfilters.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di un filtro degli indirizzi IP

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_deletereceiptfilter.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire il nome del filtro IP da eliminare. Per chiamare il metodo `deleteReceiptFilter`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Il filtro viene eliminato da Amazon SES.

```
node ses_deletereceiptfilter.js
```

Questo codice di esempio può essere trovato qui GitHub.

Utilizzo delle regole di ricezione in Amazon SES

Questo esempio di codice di Node.js illustra:

- Come creare ed eliminare le regole di ricezione.
- Come organizzare le regole di ricezione in set di regole di ricezione.

Le regole di ricezione in Amazon SES specificano cosa fare con le e-mail ricevute per indirizzi e-mail o domini di tua proprietà. Una regola di ricezione contiene una condizione e un elenco ordinato di operazioni. Se il destinatario di un'e-mail in entrata corrisponde a un destinatario specificato nelle condizioni della regola di ricezione, Amazon SES esegue le azioni specificate dalla regola di ricezione.

Per utilizzare Amazon SES come ricevitore di posta elettronica, devi avere almeno una regola di ricezione attiva. Un set di regole di ricezione è una raccolta ordinata di regole di ricezione che specificano cosa deve fare Amazon SES con la posta che riceve nei tuoi domini verificati. Per ulteriori informazioni, consulta Creazione di regole di ricezione per la ricezione di e-mail di Amazon SES e Creazione di un set di regole di ricezione per la ricezione di e-mail di Amazon SES nella Amazon Simple Email Service Developer Guide.

Lo scenario

In questo esempio, viene utilizzata una serie di moduli Node.js per inviare e-mail in diversi modi. I moduli Node.js utilizzano l'SDK per JavaScript creare e utilizzare modelli di e-mail utilizzando questi metodi della classe `AWS.SES` client:

- `createReceiptRule`
- `deleteReceiptRule`
- `createReceiptRuleSet`
- `deleteReceiptRuleSet`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni su come fornire file JSON di credenziali, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Creazione di una regola di ricezione Amazon S3

Ogni regola di ricezione per Amazon SES contiene un elenco ordinato di azioni. Questo esempio crea una regola di ricezione con un'azione Amazon S3, che consegna il messaggio di posta a un

bucket Amazon S3. Per informazioni dettagliate sulle azioni relative alle regole di ricezione, consulta le opzioni di azione nella Amazon Simple Email Service Developer Guide.

Per consentire ad Amazon SES di scrivere e-mail a un bucket Amazon S3, crea una policy per i bucket che autorizzi `PutObject` Amazon SES. Per informazioni sulla creazione di questa policy sui bucket, consulta Concedi ad Amazon SES l'autorizzazione a scrivere sul tuo bucket Amazon S3 nella Amazon Simple Email Service Developer Guide.

In questo esempio, utilizza un modulo Node.js per creare una regola di ricezione in Amazon SES per salvare i messaggi ricevuti in un bucket Amazon S3. Crea un modulo Node.js con il nome del file `ses_createreceiptrule.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto parameters per trasferire i valori necessari per creare il set di regole di ricezione. Per chiamare il metodo `createReceiptRuleSet`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES crea la regola di ricezione.

```
node ses_createreceiptrule.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di una regola di ricezione

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_deletereceiptrule.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto parameters per trasferire il nome del set di regole di ricezione da eliminare. Per chiamare il metodo `deleteReceiptRule`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES crea l'elenco delle regole di ricezione.

```
node ses_deletereceiptrule.js
```

Questo codice di esempio può essere trovato qui GitHub.

Creazione di un set di regole di ricezione

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_createreceiptruleset.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto parameters per trasferire il nome del nuovo set di regole di ricezione. Per chiamare il metodo `createReceiptRuleSet`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES crea l'elenco delle regole di ricezione.

```
node ses_createreceiptruleset.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di un set di regole di ricezione

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_deletereceiptruleset.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire il nome del set di regole di ricezione da eliminare. Per chiamare il metodo `deleeReceiptRuleSet`, crea una promessa per chiamare un oggetto del servizio Amazon SES, trasferendo i parametri. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio. Amazon SES crea l'elenco delle regole di ricezione.

```
node ses_deletereceiptruleset.js
```

Questo codice di esempio può essere trovato qui GitHub.

Esempi di servizi di notifica Amazon Simple

Amazon Simple Notification Service (Amazon SNS) è un servizio Web che coordina e gestisce la consegna o l'invio di messaggi a endpoint o client abbonati.

In Amazon SNS, esistono due tipi di clienti, editori e abbonati, noti anche come produttori e consumatori.

Gli editori comunicano in modo asincrono con i sottoscrittori producendo e inviando un messaggio a un argomento, che rappresenta un punto di accesso logico e un canale di comunicazione. Gli abbonati (server Web, indirizzi e-mail, code Amazon SQS, funzioni Lambda) utilizzano o ricevono il messaggio o la notifica tramite uno dei protocolli supportati (Amazon SQS, HTTP/S, e-mail, AWS Lambda SMS) quando sono abbonati all'argomento.

L' JavaScript API per Amazon SNS è esposta tramite. `Class: AWS.SNS`

Argomenti

Gestione degli argomenti in Amazon SNS

Questo esempio di codice di Node.js illustra:

- Come creare argomenti in Amazon SNS su cui pubblicare notifiche.
- Come eliminare argomenti creati in Amazon SNS.
- Come ottenere un elenco degli argomenti disponibili.
- Come ottenere e impostare gli attributi di argomento.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per creare, elencare ed eliminare argomenti di Amazon SNS e per gestire gli attributi degli argomenti. I moduli Node.js utilizzano l'SDK per JavaScript gestire gli argomenti utilizzando questi metodi della classe `AWS.SNS` client:

- `createTopic`
- `listTopics`
- `deleteTopic`
- `getTopicAttributes`
- `setTopicAttributes`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni su come fornire file JSON di credenziali, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Creazione di un argomento

In questo esempio, usa un modulo Node.js per creare un argomento Amazon SNS. Crea un modulo Node.js con il nome del file `sns_createtopic.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto per trasferire l'oggetto `Name` per il nuovo argomento al metodo `createTopic` della classe client `AWS.SNS`. Per chiamare il `createTopic` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa. L'oggetto `data` restituito dalla promessa contiene l'ARN dell'argomento.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_createtopic.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Elenco dei tuoi argomenti

In questo esempio, usa un modulo Node.js per elencare tutti gli argomenti di Amazon SNS. Crea un modulo Node.js con il nome del file `sns_listtopics.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto vuoto da trasferire al metodo `listTopics` della classe client `AWS.SNS`. Per chiamare il `listTopics` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa. Il `data` file restituito dalla promessa contiene una serie del tuo argomento. ARNs

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_listtopics.js
```

Questo codice di esempio può essere trovato qui GitHub.

Eliminazione di un argomento

In questo esempio, usa un modulo Node.js per eliminare un argomento di Amazon SNS. Crea un modulo Node.js con il nome del file `sns_deletetopic.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `TopicArn` dell'argomento da eliminare per passare al metodo `deleteTopic` della classe client `AWS.SNS`. Per chiamare il `deleteTopic` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_deletetopic.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Recupero degli attributi di argomento

In questo esempio, usa un modulo Node.js per recuperare gli attributi di un argomento Amazon SNS. Crea un modulo Node.js con il nome del file `sns_gettopicattributes.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `TopicArn` di un argomento da eliminare per passare al metodo `getTopicAttributes` della classe client `AWS.SNS`. Per chiamare il `getTopicAttributes` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_gettopicattributes.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Impostazione degli attributi di argomento

In questo esempio, usa un modulo Node.js per impostare gli attributi mutabili di un argomento Amazon SNS. Crea un modulo Node.js con il nome del file `sns_settopicattributes.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente i parametri per l'aggiornamento dell'attributo, inclusi il parametro `TopicArn` dell'argomento di cui desideri impostare gli attributi, il nome dell'attributo da impostare e il nuovo valore per l'attributo. È possibile impostare solo gli attributi `Policy`, `DisplayName` e `DeliveryPolicy`. Trasferisci i parametri al metodo `setTopicAttributes` della classe client `AWS.SNS`. Per chiamare il `setTopicAttributes` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_settopicattributes.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Pubblicazione di messaggi in Amazon SNS

Questo esempio di codice di Node.js illustra:

- Come pubblicare messaggi su un argomento di Amazon SNS.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per pubblicare messaggi da Amazon SNS su endpoint, e-mail o numeri di telefono tematici. I moduli Node.js utilizzano l'SDK per JavaScript inviare messaggi utilizzando questo metodo della `AWS.SNS` classe client:

- `publish`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni su come fornire file JSON di credenziali, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Pubblicazione di un messaggio su un argomento di Amazon SNS

In questo esempio, usa un modulo Node.js per pubblicare un messaggio su un argomento di Amazon SNS. Crea un modulo Node.js con il nome del file `sns_publishtotopic.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente i parametri per la pubblicazione di un messaggio, inclusi il testo del messaggio e l'ARN dell'argomento Amazon SNS. Per i dettagli sugli attributi SMS disponibili, consulta Set. SMSAttributes

Trasferisci i parametri al metodo `publish` della classe client `AWS.SNS`. Crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci la risposta nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_publishtotopic.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Gestione degli abbonamenti in Amazon SNS

Questo esempio di codice di Node.js illustra:

- Come elencare tutti gli abbonamenti a un argomento di Amazon SNS.
- Come sottoscrivere un indirizzo e-mail, un endpoint dell'applicazione o una AWS Lambda funzione a un argomento di Amazon SNS.
- Come annullare l'iscrizione agli argomenti di Amazon SNS.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per pubblicare messaggi di notifica su argomenti di Amazon SNS. I moduli Node.js utilizzano l'SDK per JavaScript gestire gli argomenti utilizzando questi metodi della classe `AWS.SNS` client:

- `subscribe`
- `confirmSubscription`
- `listSubscriptionsByTopic`
- `unsubscribe`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni su come fornire file JSON di credenziali, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Elenco di sottoscrizioni a un argomento

In questo esempio, usa un modulo Node.js per elencare tutte le sottoscrizioni a un argomento di Amazon SNS. Crea un modulo Node.js con il nome del file `sns_listsubscriptions.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `TopicArn` per l'argomento di cui si desideri elencare le sottoscrizioni. Trasferisci i parametri al metodo `listSubscriptionsByTopic` della classe client `AWS.SNS`. Per chiamare il `listSubscriptionsByTopic` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_listsubscriptions.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Sottoscrizione di un indirizzo e-mail a un argomento

In questo esempio, usa un modulo Node.js per sottoscrivere un indirizzo e-mail in modo che riceva messaggi e-mail SMTP da un argomento di Amazon SNS. Crea un modulo Node.js con il nome del file `sns_subscribeemail.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `Protocol` per specificare il protocollo `email`, il parametro `TopicArn` per l'argomento a cui effettuare la sottoscrizione e un indirizzo e-mail come `Endpoint` del messaggio. Trasferisci i parametri al metodo `subscribe` della classe client `AWS.SNS`. Puoi utilizzare il `subscribe` metodo per sottoscrivere diversi endpoint a un argomento Amazon SNS, a seconda dei valori utilizzati per i parametri passati, come mostreranno altri esempi in questo argomento.

Per chiamare il `subscribe` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_subscribeemail.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Sottoscrizione di un endpoint di applicazione a un argomento

In questo esempio, usa un modulo Node.js per sottoscrivere un endpoint di applicazione mobile in modo che riceva notifiche da un argomento di Amazon SNS. Crea un modulo Node.js con il nome del file `sns_subscribeapp.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `Protocol` per specificare il protocollo `application`, il parametro `TopicArn` per l'argomento a cui effettuare la sottoscrizione e l'ARN di un endpoint di applicazione mobile per il parametro `Endpoint`. Trasferisci i parametri al metodo `subscribe` della classe client `AWS.SNS`.

Per chiamare il `subscribe` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_subscribeapp.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Sottoscrizione di una funzione Lambda a un argomento

In questo esempio, usa un modulo Node.js per sottoscrivere una AWS Lambda funzione in modo che riceva notifiche da un argomento di Amazon SNS. Crea un modulo Node.js con il nome del file `sns_subscribelambda.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il `Protocol` parametro, specificando il `lambda` protocollo, `TopicArn` l'argomento a cui sottoscrivere e l'ARN di AWS Lambda una funzione come `Endpoint` parametro. Trasferisci i parametri al metodo `subscribe` della classe client `AWS.SNS`.

Per chiamare il `subscribe` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_subscribelambda.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Annullamento della sottoscrizione a un argomento

In questo esempio, usa un modulo Node.js per annullare l'iscrizione a un argomento Amazon SNS. Crea un modulo Node.js con il nome del file `sns_unsubscribe.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `SubscriptionArn`, che specifica l'ARN della sottoscrizione da annullare. Trasferisci i parametri al metodo `unsubscribe` della classe client `AWS.SNS`.

Per chiamare il `unsubscribe` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_unsubscribe.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Invio di messaggi SMS con Amazon SNS

Questo esempio di codice di Node.js illustra:

- Come ottenere e impostare le preferenze di messaggistica SMS per Amazon SNS.
- Come controllare un numero di telefono per verificare se è stata disattivata la ricezione di messaggi SMS.
- Come ottenere un elenco di numeri di telefono per cui è stata disattivata la ricezione di messaggi SMS.
- Come inviare un messaggio SMS.

Lo scenario

Puoi utilizzare Amazon SNS; per inviare messaggi SMS a dispositivi abilitati. Puoi inviare un messaggio direttamente a un numero di telefono oppure inviarlo a più numeri contemporaneamente sottoscrivendo quei numeri a un argomento e inviando il messaggio all'argomento.

In questo esempio, utilizzi una serie di moduli Node.js per pubblicare messaggi di testo SMS da Amazon SNS a dispositivi abilitati all'invio di SMS. I moduli Node.js utilizzano l'SDK per JavaScript pubblicare messaggi SMS utilizzando questi metodi della classe client: `AWS.SNS`

- `getSMSAttributes`
- `setSMSAttributes`
- `checkIfPhoneNumberIsOptedOut`
- `listPhoneNumbersOptedOut`
- `publish`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni su come fornire file JSON di credenziali, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Recupero degli attributi SMS

Usa Amazon SNS per specificare le preferenze per la messaggistica SMS, ad esempio il modo in cui le consegne sono ottimizzate (in termini di costi o per una consegna affidabile), il limite di spesa mensile, il modo in cui vengono registrate le consegne dei messaggi e se abbonarsi ai report giornalieri sull'utilizzo degli SMS. Queste preferenze vengono recuperate e impostate come attributi SMS per Amazon SNS.

In questo esempio, usa un modulo Node.js per ottenere gli attributi SMS correnti in Amazon SNS. Crea un modulo Node.js con il nome del file `sns_getsmstype.js`. Configura l'SDK come mostrato in precedenza. Crea un oggetto contenente i parametri per ottenere attributi SMS, inclusi i nomi dei singoli attributi da recuperare. Per i dettagli sugli attributi SMS disponibili, consulta Set SMSAttributes in the Amazon Simple Notification Service API Reference.

Questo esempio recupera l'attributo `DefaultSMSType`, che controlla se i messaggi SMS vengono inviati come `Promotional` per ottimizzare il recapito dei messaggi e permettere di contenere i costi, oppure come `Transactional` per ottimizzare il recapito dei messaggi e ottenere la massima affidabilità. Trasferisci i parametri al metodo `setTopicAttributes` della classe client `AWS.SNS`. Per chiamare il `getSMSAttributes` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_getsmstype.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Impostazione degli attributi SMS

In questo esempio, usa un modulo Node.js per ottenere gli attributi SMS correnti in Amazon SNS. Crea un modulo Node.js con il nome del file `sns_setsmstype.js`. Configura l'SDK come mostrato in precedenza. Crea un oggetto contenente i parametri per impostare gli attributi SMS, inclusi i nomi dei singoli attributi da impostare e i valori da impostare per ciascuno di essi. Per i dettagli sugli attributi SMS disponibili, consulta Set SMSAttributes in the Amazon Simple Notification Service API Reference.

Questo esempio imposta l'attributo `DefaultSMSType` su `Transactional`, ottimizzando il recapito dei messaggi per ottenere la massima affidabilità. Trasferisci i parametri al metodo `setTopicAttributes` della classe client `AWS.SNS`. Per chiamare il `getSMSAttributes` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_setsmstype.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Controllare se un numero di telefono è stato disattivato

In questo esempio, utilizza un modulo Node.js per controllare un numero di telefono per verificare se è stata disattivata la ricezione di messaggi SMS. Crea un modulo Node.js con il nome del file `sns_checkphoneoptout.js`. Configura l'SDK come mostrato in precedenza. Crea un oggetto contenente il numero di telefono da controllare come parametro.

Questo esempio imposta il parametro `PhoneNumber` per specificare il numero di telefono da verificare. Trasferisci l'oggetto al metodo `checkIfPhoneNumberIsOptedOut` della classe client `AWS.SNS`: Per chiamare il `checkIfPhoneNumberIsOptedOut` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_checkphoneoptout.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Elenco di numeri di telefono disattivati

In questo esempio, utilizza un modulo Node.js per ottenere un elenco di numeri di telefono per cui è stata disattivata la ricezione di messaggi SMS. Crea un modulo Node.js con il nome del file `sns_listnumbersoptedout.js`. Configura l'SDK come mostrato in precedenza. Crea un oggetto vuoto come parametro.

Trasferisci l'oggetto al metodo `listPhoneNumbersOptedOut` della classe client `AWS.SNS`: Per chiamare il `listPhoneNumbersOptedOut` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_listnumbersoptedout.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Pubblicazione di un messaggio SMS

In questo esempio, utilizza un modulo Node.js per inviare un messaggio SMS a un numero di telefono. Crea un modulo Node.js con il nome del file `sns_publishsms.js`. Configura l'SDK come mostrato in precedenza. Crea un oggetto contenente i parametri `Message` e `PhoneNumber`.

Quando invii un SMS, ricorda di specificare il numero di telefono utilizzando il formato E.164. E.164 è uno standard per la struttura del numero di telefono utilizzato per le telecomunicazioni internazionali. I numeri di telefono che seguono questo formato possono avere un massimo di 15 cifre e sono preceduti dal segno più (+) e dal prefisso internazionale. Ad esempio, un numero di telefono statunitense in formato E.164 apparirebbe come XXX5550100 +1001.

Questo esempio imposta il parametro `PhoneNumber` per specificare il numero di telefono a cui inviare il messaggio. Trasferisci l'oggetto al metodo `publish` della classe client `AWS.SNS`: Per chiamare il `publish` metodo, crea una promessa per richiamare un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Quindi gestisci l'oggetto `response` nel callback della promessa.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sns_publishsms.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Esempi di Amazon SQS

Amazon Simple Queue Service (Amazon SQS) è un servizio di accodamento dei messaggi veloce, affidabile, scalabile e completamente gestito. Amazon SQS consente di disaccoppiare i componenti di un'applicazione cloud. Amazon SQS include code standard con throughput ed at-least-once elaborazione elevati e code FIFO che forniscono consegne FIFO (first-in, first-out) ed elaborazione Exactly-Once.

L' JavaScript API per Amazon SQS è esposta tramite la classe `AWS.SQS` client. Per ulteriori informazioni sull'utilizzo della classe client Amazon SQS, consulta il riferimento `Class: AWS.SQS`all'API.

Argomenti

- Utilizzo delle code in Amazon SQS
- Invio e ricezione di messaggi in Amazon SQS
- Gestione del timeout visibilità in Amazon SQS
- Abilitazione del polling lungo in Amazon SQS
- Utilizzo delle code DLQ in Amazon SQS

Utilizzo delle code in Amazon SQS

Questo esempio di codice di Node.js illustra:

- Come ottenere un elenco di tutte le code di messaggi
- Come ottenere l'URL per una determinata coda
- Come creare ed eliminare le code

Informazioni sull'esempio

In questo esempio, viene utilizzata una serie di moduli Node.js per lavorare con le code. I moduli Node.js utilizzano l'SDK per consentire JavaScript alle code di chiamare i seguenti metodi della `AWS.SQS` classe client:

- `listQueues`
- `createQueue`
- `getQueueUrl`
- `deleteQueue`

Per ulteriori informazioni sui messaggi Amazon SQS, consulta How Queues Work nella Amazon Simple Queue Service Developer Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Elenco di code

Crea un modulo Node.js con il nome del file `sqs_listqueues.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon SQS, crea un oggetto di `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per elencare le code, che per impostazione predefinita è un oggetto vuoto. Chiama il metodo `listQueues` per recuperare l'elenco di code. Il callback restituisce tutte URLs le code.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_listqueues.js
```

Questo codice di esempio può essere trovato qui. GitHub

Creazione di una coda

Crea un modulo Node.js con il nome del file `sqs_createqueue.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon SQS, crea un oggetto di `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per elencare le code, incluso il nome della coda creata. I parametri possono anche contenere attributi per la coda, come i secondi di ritardo della consegna del messaggio o i secondi di conservazione di un messaggio ricevuto. Chiama il metodo `createQueue`. Il callback restituisce l'URL della coda creata.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_createqueue.js
```

Questo codice di esempio può essere trovato qui. GitHub

Recupero dell'URL di una coda

Crea un modulo Node.js con il nome del file `sqs_getqueueurl.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon SQS, crea un oggetto di `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per elencare le code, incluso il nome della coda di cui desideri l'URL. Chiama il metodo `getQueueUrl`. Il callback restituisce l'URL della coda specificata.

```
// 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);
  }
```

```
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_getqueueurl.js
```

Questo codice di esempio può essere trovato qui. GitHub

Eliminazione di una coda

Crea un modulo Node.js con il nome del file `sqs_deletequeue.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon SQS, crea un oggetto di `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per eliminare una coda, che consiste nell'URL della coda da eliminare. Chiama il metodo `deleteQueue`.

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

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_deletequeue.js
```

Questo codice di esempio può essere trovato qui. GitHub

Invio e ricezione di messaggi in Amazon SQS

Questo esempio di codice di Node.js illustra:

- Come inviare messaggi in una coda.
- Come ricevere messaggi in una coda.
- Come eliminare messaggi in una coda.

Lo scenario

In questo esempio, viene utilizzata una serie di moduli Node.js per inviare e ricevere messaggi. I moduli Node.js utilizzano l'SDK JavaScript per inviare e ricevere messaggi utilizzando questi metodi della classe `AWS.SQS` client:

- `sendMessage`
- `receiveMessage`
- `deleteMessage`

Per ulteriori informazioni sui messaggi Amazon SQS, consulta Invio di un messaggio a una coda Amazon SQS e Ricezione ed eliminazione di un messaggio da una coda Amazon SQS nella Amazon Simple Queue Service Developer Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Creare una coda Amazon SQS. Per un esempio di creazione di una coda, consulta Utilizzo delle code in Amazon SQS.

Invio di un messaggio a una coda

Crea un modulo Node.js con il nome del file `sqs_sendmessage.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon SQS, crea un oggetto di `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per il tuo messaggio, incluso l'URL della coda a cui inviare il messaggio. In questo esempio, il messaggio fornisce informazioni su un libro presente in una classifica di romanzi best seller, comprendente il titolo, l'autore e il numero di settimane in classifica.

Chiama il metodo `sendMessage`. Il callback restituisce l'ID univoco del messaggio.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_sendmessage.js
```

Questo codice di esempio può essere trovato qui. GitHub

Ricezione ed eliminazione di messaggi da una coda

Crea un modulo Node.js con il nome del file `sqs_receivemessage.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon SQS, crea un oggetto di `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per il tuo messaggio, incluso l'URL della coda da cui ricevere il messaggio. In questo esempio, i parametri specificano la ricezione di tutti gli attributi del messaggio, nonché la ricezione di non più di 10 messaggi.

Chiama il metodo `receiveMessage`. Il callback restituisce una matrice di oggetti `Message` da cui è possibile recuperare il valore `ReceiptHandle` per ogni messaggio utilizzato in seguito per eliminare il messaggio. Crea un altro oggetto JSON contenente i parametri necessari per eliminare il messaggio, ovvero l'URL della coda e il valore `ReceiptHandle`. Chiama il metodo `deleteMessage` per eliminare il messaggio ricevuto.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_receivemessage.js
```

Questo codice di esempio può essere trovato qui. GitHub

Gestione del timeout visibilità in Amazon SQS

Questo esempio di codice di Node.js illustra:

- Come specificare l'intervallo di tempo durante il quale i messaggi ricevuti da una coda non sono visibili.

Lo scenario

In questo esempio, viene utilizzato un modulo Node.js per gestire il timeout visibilità. Il modulo Node.js utilizza l'SDK per JavaScript gestire il timeout di visibilità utilizzando questo metodo della classe `AWS.SQS` client:

- `changeMessageVisibility`

Per ulteriori informazioni sul timeout di visibilità di Amazon SQS, consulta Visibility Timeout nella Amazon Simple Queue Service Developer Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Creare una coda Amazon SQS. Per un esempio di creazione di una coda, consulta Utilizzo delle code in Amazon SQS.
- Invio di un messaggio alla coda. Per un esempio di invio di un messaggio a una coda, consulta Invio e ricezione di messaggi in Amazon SQS.

Modifica del timeout visibilità

Crea un modulo Node.js con il nome del file `sqs_changingvisibility.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon Simple Queue Service, crea un oggetto `AWS.SQS` servizio. Ricevere il messaggio dalla coda.

Alla ricezione del messaggio dalla coda, crea un oggetto JSON contenente i parametri necessari per impostare il timeout, inclusi l'URL della coda che contiene il messaggio, il valore `ReceiptHandle` restituito quando il messaggio è stato ricevuto e il nuovo timeout in secondi. Chiama il metodo `changeMessageVisibility`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
```

```
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_changingvisibility.js
```

Questo codice di esempio può essere trovato qui. GitHub

Abilitazione del polling lungo in Amazon SQS

Questo esempio di codice di Node.js illustra:

- Come abilitare il long polling per una nuova coda
- Come abilitare il long polling per una coda esistente
- Come abilitare il long polling alla ricezione di un messaggio

Lo scenario

Il polling prolungato riduce il numero di risposte vuote consentendo ad Amazon SQS di attendere un periodo di tempo specificato affinché un messaggio diventi disponibile nella coda prima di inviare una risposta. Inoltre, il polling lungo elimina le risposte vuote false creando delle query per tutti i server invece di effettuarne il campionamento. Per abilitare il polling lungo, occorre specificare un tempo di attesa diverso da zero per i messaggi ricevuti. A tale scopo puoi impostare il parametro `ReceiveMessageWaitTimeSeconds` di una coda o impostare il parametro `WaitTimeSeconds` su un messaggio quando viene ricevuto.

In questo esempio, viene utilizzata una serie di moduli Node.js per abilitare il long polling. I moduli Node.js utilizzano l'SDK per JavaScript abilitare il polling lungo utilizzando questi metodi della classe client: `AWS.SQS`

- `setQueueAttributes`
- `receiveMessage`
- `createQueue`

Per ulteriori informazioni sul long polling di Amazon SQS, consulta Long Polling nella Amazon Simple Queue Service Developer Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.

Abilitazione del long polling alla creazione di una coda

Crea un modulo Node.js con il nome del file `sqs_longpolling_createqueue.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon SQS, crea un oggetto di `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per creare una coda, incluso un valore diverso da zero per il parametro `ReceiveMessageWaitTimeSeconds`. Chiama il metodo `createQueue`. Il long polling viene quindi abilitato per la coda.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_longpolling_createqueue.js
```

Questo codice di esempio può essere trovato qui. GitHub

Abilitazione del long polling su una coda esistente

Crea un modulo Node.js con il nome del file `sqs_longpolling_existingqueue.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon Simple Queue Service, crea un oggetto `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per impostare gli attributi della coda, inclusi un valore diverso da zero per il parametro `ReceiveMessageWaitTimeSeconds` e l'URL della coda. Chiama il metodo `setQueueAttributes`. Il long polling viene quindi abilitato per la coda.

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_longpolling_existingqueue.js
```

Questo codice di esempio può essere trovato qui. GitHub

Abilitazione del long polling alla ricezione del messaggio

Crea un modulo Node.js con il nome del file `sqs_longpolling_receivemessage.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon Simple Queue Service, crea un oggetto `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per ricevere messaggi, inclusi un valore diverso da zero per il parametro `WaitTimeSeconds` e l'URL della coda. Chiama il metodo `receiveMessage`.

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

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_longpolling_receivemessage.js
```

Questo codice di esempio può essere trovato qui. GitHub

Utilizzo delle code DLQ in Amazon SQS

Questo esempio di codice di Node.js illustra:

- Come utilizzare una coda per ricevere e mantenere i messaggi da altre code che le code non sono in grado di elaborare.

Lo scenario

Una coda DLQ è una coda a cui altre code (origini) possono mirare per i messaggi che non possono essere elaborati correttamente. Puoi riservare e isolare questi messaggi nella coda DLQ per determinare perché l'elaborazione non è riuscita. Devi configurare singolarmente ogni coda di origine che invia messaggi a una coda DLQ. Code multiple possono mirare a una singola coda DLQ.

In questo esempio, viene utilizzato un modulo Node.js per instradare messaggi a una coda DLQ. Il modulo Node.js utilizza l'SDK per JavaScript utilizzare le code di lettere morte utilizzando questo metodo della classe `AWS.SQS` client:

- `setQueueAttributes`

Per ulteriori informazioni sulle code di lettere morte di Amazon SQS, consulta Using Amazon SQS Dead Letter Queues nella Amazon Simple Queue Service Developer Guide.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Installa Node.js. Per ulteriori informazioni sull'installazione di Node.js, consulta il sito Web Node.js.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file delle credenziali condiviso, consulta Caricamento delle credenziali su Node.js dal file delle credenziali condiviso.
- Crea una coda Amazon SQS che funga da coda di lettere morte. Per un esempio di creazione di una coda, consulta Utilizzo delle code in Amazon SQS.

Configurazione delle code di origine

Una volta creata una coda che si comporta come coda DLQ, è necessario configurare le altre code in modo che instradino i messaggi non elaborati alla coda DLQ. Per eseguire questa operazione, specificare una policy di reindirizzamento che identifichi la coda da utilizzare come coda DLQ e il numero massimo di singoli messaggi da ricevere prima che siano instradati alla coda DLQ.

Crea un modulo Node.js con il nome del file `sqs_deadletterqueue.js`. Assicurati di configurare il kit SDK come mostrato in precedenza. Per accedere ad Amazon SQS, crea un oggetto di `AWS.SQS` servizio. Crea un oggetto JSON contenente i parametri necessari per aggiornare gli attributi della coda, incluso il parametro `RedrivePolicy` che specifica sia l'Amazon Resource Name (ARN) della coda DLQ sia il valore di `maxReceiveCount`. Specifica inoltre l'URL della coda di origine che desideri configurare. Chiama il metodo `setQueueAttributes`.

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

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  Attributes: {
    RedrivePolicy:
      '{"deadLetterTargetArn":"DEAD_LETTER_QUEUE_ARN","maxReceiveCount":"10"}',
  },
  QueueUrl: "SOURCE_QUEUE_URL",
};

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

Digita la seguente riga di comando per eseguire l'esempio.

```
node sqs_deadletterqueue.js
```

Questo codice di esempio può essere trovato qui. GitHub

Esercitazioni

I seguenti tutorial mostrano come eseguire diverse attività relative all'utilizzo dell' AWS SDK per JavaScript.

Argomenti

Tutorial: configurazione di Node.js su un'istanza Amazon EC2

Uno scenario comune per l'utilizzo di Node.js con l'SDK per JavaScript consiste nel configurare ed eseguire un'applicazione Web Node.js su un'istanza Amazon Elastic Compute Cloud (Amazon EC2). In questo tutorial, creerai un'istanza Linux, ti connetterai a essa tramite SSH, quindi installerai Node.js per l'esecuzione su tale istanza.

Prerequisiti

Questo tutorial presuppone che tu abbia già avviato una nuova istanza Linux con un nome DNS pubblico raggiungibile da Internet e alla quale è possibile connettersi tramite SSH. Per ulteriori informazioni, consulta la Fase 1: Avvio di un'istanza nella Guida per l'utente di Amazon EC2.

> **Important**
>
> Usa Amazon Linux 2023 Amazon Machine Image (AMI) per lanciare una nuova istanza Amazon EC2.

È inoltre necessario aver configurato il gruppo di sicurezza per consentire le connessioni SSH (porta 22), HTTP (porta 80) e HTTPS (porta 443). Per ulteriori informazioni su questi prerequisiti, consulta Configurazione con Amazon Amazon EC2 nella Guida per l'utente di Amazon EC2.

Procedura

La procedura seguente ti consente di installare Node.js su un'istanza Amazon Linux. Puoi utilizzare questo server per l'hosting di un'applicazione Web Node.js.

Per configurare Node.js sulla tua istanza Linux

1. Connettersi all'istanza Linux come `ec2-user` tramite SSH.
2. Installare il gestore delle versioni del nodo (nvm) digitando quanto segue nella riga di comando.

   > **Warning**
   >
   > AWS non controlla il seguente codice. Prima di eseguirlo, assicurati di verificarne l'autenticità e l'integrità. Ulteriori informazioni su questo codice sono disponibili nel repository nvm. GitHub

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
   ```

   Verrà utilizzato nvm per installare Node.js perché è in grado di installare più versioni di Node.js e consente di passare dall'una all'altra.
3. Carica `nvm` digitando quanto segue nella riga di comando.

   ```
   source ~/.bashrc
   ```
4. Usa nvm per installare l'ultima versione LTS di Node.js digitando quanto segue nella riga di comando.

   ```
   nvm install --lts
   ```

   L'installazione di Node.js installa anche il Node Package Manager (npm), quindi puoi installare moduli aggiuntivi secondo necessità.
5. Verificare che Node.js sia installato e correttamente in esecuzione digitando quanto segue nella riga di comando.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Viene visualizzato il seguente messaggio che mostra la versione di Node.js in esecuzione.

   ```
   Running Node.js VERSION
   ```

> **Note**
>
> L'installazione del nodo si applica solo alla sessione corrente di Amazon EC2. Se riavvii la sessione CLI, devi usare nvm per abilitare la versione del nodo installata. Se l'istanza viene terminata, è necessario installare nuovamente il nodo. L'alternativa è creare un'Amazon Machine Image (AMI) dell'istanza Amazon EC2 una volta ottenuta la configurazione che desideri conservare, come descritto nel seguente argomento.

Creazione di un'Amazon Machine Image

Dopo aver installato Node.js su un'istanza Amazon EC2, puoi creare un'Amazon Machine Image (AMI) da quell'istanza. La creazione di un'AMI semplifica il provisioning di più istanze Amazon EC2 con la stessa installazione di Node.js. Per ulteriori informazioni sulla creazione di un'AMI da un'istanza esistente, consulta Creazione di un'AMI Linux supportata da Amazon EBS nella Guida per l'utente di Amazon EC2.

Risorse correlate

Per ulteriori informazioni sui comandi e sul software utilizzati in questo argomento, consulta le pagine Web seguenti:

- node version manager (nvm): consulta nvm repo on. GitHub
- gestore di pacchetti nodo (npm): consulta il sito Web npm.

JavaScript Riferimento API

Gli argomenti di riferimento sulle API per la versione più recente dell'SDK JavaScript sono disponibili all'indirizzo:

AWS SDK per JavaScript Guida di riferimento alle API.

Log delle modifiche SDK attivo GitHub

Il changelog per le versioni a partire dalla 2.4.8 è disponibile all'indirizzo:

Registro delle modifiche.

Migrare alla v3 di AWS SDK per JavaScript

La AWS SDK per JavaScript versione 3 è una riscrittura importante della versione 2. Per ulteriori informazioni sulla migrazione alla versione 3, consulta Migrare dalla versione 2.x alla 3.x AWS SDK per JavaScript nella Developer Guide v3.AWS SDK per JavaScript

Sicurezza per questo AWS prodotto o servizio

La sicurezza cloud di Amazon Web Services (AWS) è la priorità più alta. In qualità di cliente AWS , è possibile trarre vantaggio da un'architettura di data center e di rete progettata per soddisfare i requisiti delle organizzazioni più esigenti a livello di sicurezza. La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud.

Security of the Cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce tutti i servizi offerti nel AWS Cloud e della fornitura di servizi che è possibile utilizzare in modo sicuro. La nostra responsabilità in AWS materia di sicurezza è la massima priorità e l'efficacia della nostra sicurezza viene regolarmente testata e verificata da revisori di terze parti nell'ambito dei Programmi di AWS conformità.

Sicurezza nel cloud: la responsabilità dell'utente è determinata dal AWS servizio utilizzato e da altri fattori, tra cui la sensibilità dei dati, i requisiti dell'organizzazione e le leggi e i regolamenti applicabili.

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Argomenti

Protezione dei dati in questo AWS prodotto o servizio

Il modello di responsabilità AWS condivisa di si applica alla protezione dei dati in questo AWS prodotto o servizio. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione

della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per maggiori informazioni sulla privacy dei dati, consulta le Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al AWS Modello di responsabilità condivisa e GDPR nel AWS Blog sulla sicurezza.

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- SSL/TLS Da utilizzare per comunicare con AWS le risorse. È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con AWS CloudTrail. Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta Lavorare con i CloudTrail percorsi nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-3.

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando si lavora con questo AWS prodotto o servizio o altro Servizi AWS utilizzando la console, l'API o. AWS CLI AWS SDKs I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando si fornisce un URL a un server esterno, suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la richiesta al server.

Identity and Access Management

AWS Identity and Access Management (IAM) è un software Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori

IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle autorizzazioni) a utilizzare le risorse. AWS IAM è uno Servizio AWS strumento che puoi utilizzare senza costi aggiuntivi.

Argomenti

Destinatari

Il modo in cui usi AWS Identity and Access Management (IAM) varia a seconda del lavoro che AWS svolgi.

Utente del servizio: se lo utilizzi Servizi AWS per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più AWS funzionalità per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell’accesso consente di richiedere le autorizzazioni corrette all’amministratore. Se non riesci ad accedere a una funzionalità di AWS, consulta Risoluzione dei problemi di AWS identità e accesso o consulta la guida per l'utente della funzionalità Servizio AWS che stai utilizzando.

Amministratore del servizio: se sei responsabile delle AWS risorse della tua azienda, probabilmente hai pieno accesso a AWS. È tuo compito determinare a quali AWS funzionalità e risorse devono accedere gli utenti del servizio. Devi quindi inviare le richieste all’amministratore IAM per modificare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per saperne di più su come la tua azienda può utilizzare IAM con AWS, consulta la guida per l'utente del Servizio AWS software che stai utilizzando.

Amministratore IAM: un amministratore IAM potrebbe essere interessato a ottenere dei dettagli su come scrivere policy per gestire l'accesso a AWS. Per visualizzare esempi di policy AWS basate sull'identità che puoi utilizzare in IAM, consulta la guida per l'utente di quella Servizio AWS che stai utilizzando.

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi autenticarti come utente IAM o assumendo un ruolo IAM. Utente root dell'account AWS

Puoi accedere come identità federata utilizzando credenziali provenienti da una fonte di identità come AWS IAM Identity Center (IAM Identity Center), autenticazione Single Sign-On o credenziali. Google/ Facebook Per ulteriori informazioni sull'accesso, consulta Come accedere all' Account AWS nella Guida per l'utente di Accedi ad AWS .

Per l'accesso programmatico, AWS fornisce un SDK e una CLI per firmare crittograficamente le richieste. Per ulteriori informazioni, consulta AWS Signature Version 4 per le richieste API nella Guida per l'utente di IAM.

Account AWS utente root

Quando si crea un Account AWS, si inizia con un'identità di accesso denominata utente Account AWS root che ha accesso completo a tutte Servizi AWS le risorse. Consigliamo vivamente di non utilizzare l'utente root per le attività quotidiane. Per le attività che richiedono le credenziali dell'utente root, consulta Attività che richiedono le credenziali dell'utente root nella Guida per l'utente IAM.

Identità federata

Come procedura ottimale, richiedi agli utenti umani di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente della directory aziendale, del provider di identità Web o Directory Service che accede Servizi AWS utilizzando le credenziali di una fonte di identità. Le identità federate assumono ruoli che forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, si consiglia di utilizzare AWS IAM Identity Center. Per ulteriori informazioni, consulta Che cos'è il Centro identità IAM? nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un utente IAM è una identità che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ti consigliamo di utilizzare credenziali temporanee invece di utenti IAM con credenziali a lungo termine. Per ulteriori informazioni, consulta Richiedere agli utenti umani di utilizzare la

federazione con un provider di identità per accedere AWS utilizzando credenziali temporanee nella Guida per l'utente IAM.

Un gruppo IAM specifica una raccolta di utenti IAM e semplifica la gestione delle autorizzazioni per gestire gruppi di utenti di grandi dimensioni. Per ulteriori informazioni, consulta Casi d'uso per utenti IAM nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità con autorizzazioni specifiche che fornisce credenziali temporanee. Puoi assumere un ruolo passando da un ruolo utente a un ruolo IAM (console) o chiamando un'operazione AWS CLI o AWS API. Per ulteriori informazioni, consulta Metodi per assumere un ruolo nella Guida per l'utente di IAM.

I ruoli IAM sono utili per l'accesso degli utenti federati, le autorizzazioni utente IAM temporanee, l'accesso multi-account, l'accesso multi-servizio e le applicazioni in esecuzione su Amazon EC2. Per maggiori informazioni, consultare Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Gestione dell'accesso tramite policy

Puoi controllare l'accesso AWS creando policy e associandole a AWS identità o risorse. Una policy definisce le autorizzazioni quando è associata a un'identità o a una risorsa. AWS valuta queste politiche quando un preside effettua una richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per maggiori informazioni sui documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Utilizzando le policy, gli amministratori specificano chi ha accesso a cosa definendo quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Un amministratore IAM crea le policy IAM e le aggiunge ai ruoli, che gli utenti possono quindi assumere. Le policy IAM definiscono le autorizzazioni indipendentemente dal metodo utilizzato per eseguirle.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile collegare a un'identità (utente, gruppo o ruolo). Tali policy controllano le operazioni autorizzate per l'identità, nonché le risorse e le condizioni in cui possono essere eseguite. Per informazioni su come

creare una policy basata su identità, consultare Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Le policy basate su identità possono essere policy in linea (con embedding direttamente in una singola identità) o policy gestite (policy autonome collegate a più identità). Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scegliere tra policy gestite e policy in linea nella Guida per l'utente di IAM.

Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi includono le policy di trust dei ruoli IAM e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. In una policy basata sulle risorse è obbligatorio specificare un'entità principale.

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Elenchi di controllo degli accessi () ACLs

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano. AWS WAF ACLs Per ulteriori informazioni ACLs, consulta la panoramica della lista di controllo degli accessi (ACL) nella Amazon Simple Storage Service Developer Guide.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi che possono impostare le autorizzazioni massime concesse dai tipi di policy più comuni:

- Limiti delle autorizzazioni: imposta il numero massimo di autorizzazioni che una policy basata su identità ha la possibilità di concedere a un'entità IAM. Per ulteriori informazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente di IAM.
- Politiche di controllo del servizio (SCPs): specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa in. AWS Organizations Per ulteriori informazioni, consultare Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations .

- Politiche di controllo delle risorse (RCPs): imposta le autorizzazioni massime disponibili per le risorse nei tuoi account. Per ulteriori informazioni, consulta Politiche di controllo delle risorse (RCPs) nella Guida per l'AWS Organizations utente.
- Policy di sessione: policy avanzate passate come parametro quando si crea una sessione temporanea per un ruolo o un utente federato. Per maggiori informazioni, consultare Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando a una richiesta si applicano più tipi di policy, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella IAM User Guide.

Come Servizi AWS lavorare con IAM

Per avere una visione di alto livello di come Servizi AWS funziona la maggior parte delle funzionalità IAM, consulta AWS i servizi che funzionano con IAM nella IAM User Guide.

Per scoprire come utilizzare uno specifico Servizio AWS con IAM, consulta la sezione sulla sicurezza della Guida per l'utente del servizio pertinente.

Risoluzione dei problemi di AWS identità e accesso

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con un AWS IAM.

Argomenti

- Non sono autorizzato a eseguire alcuna azione in AWS
- Non sono autorizzato a eseguire iam: PassRole
- Voglio consentire a persone esterne a me di accedere Account AWS alle mie AWS risorse

Non sono autorizzato a eseguire alcuna azione in AWS

Se ricevi un errore che indica che non sei autorizzato a eseguire un'operazione, le tue policy devono essere aggiornate per poter eseguire l'operazione.

L'errore di esempio seguente si verifica quando l'utente IAM `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa *`my-example-widget`* fittizia ma non dispone di autorizzazioni `awes:`*`GetWidget`* fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In questo caso, la policy per l'utente `mateojackson` deve essere aggiornata per consentire l'accesso alla risorsa *`my-example-widget`* utilizzando l'azione `awes:`*`GetWidget`*.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un errore che indica che non sei autorizzato a eseguire l'operazione `iam:PassRole`, le tue policy devono essere aggiornate per poter passare un ruolo a AWS.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

L'errore di esempio seguente si verifica quando un utente IAM denominato `marymajor` cerca di utilizzare la console per eseguire un'operazione in AWS. Tuttavia, l'operazione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per trasmettere il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie AWS risorse

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo.

Per i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per concedere alle persone l'accesso alle tue risorse.

Per maggiori informazioni, consulta gli argomenti seguenti:

- Per sapere se AWS supporta queste funzionalità, consulta. Come Servizi AWS lavorare con IAM
- Per scoprire come fornire l'accesso alle tue risorse attraverso Account AWS le risorse di tua proprietà, consulta Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà nella IAM User Guide.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta Fornire l'accesso a soggetti Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso a utenti autenticati esternamente (federazione delle identità) nella Guida per l'utente IAM.
- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente di IAM.

Convalida della conformità per questo AWS prodotto o servizio

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione Scope by Compliance Program Servizi AWS e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di AWS conformità Programmi di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in AWS Artifact .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. Per ulteriori informazioni sulla responsabilità di conformità durante l'utilizzo Servizi AWS, consulta AWS la documentazione sulla sicurezza.

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Resilienza per questo AWS prodotto o servizio

L'infrastruttura AWS globale è costruita attorno a zone Regioni AWS di disponibilità.

Regioni AWS forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti.

Con le zone di disponibilità è possibile progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, vedere Global Infrastructure.AWS

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Sicurezza dell'infrastruttura per questo AWS prodotto o servizio

Questo AWS prodotto o servizio utilizza servizi gestiti ed è pertanto protetto dalla sicurezza di rete AWS globale. Per informazioni sui servizi AWS di sicurezza e su come AWS protegge l'infrastruttura, consulta AWS Cloud Security. Per progettare il tuo AWS ambiente utilizzando le migliori pratiche per la sicurezza dell'infrastruttura, vedi Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Utilizzate chiamate API AWS pubblicate per accedere a questo AWS Prodotto o Servizio attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. In alternativa è possibile utilizzare AWS Security Token Service (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Applicazione di una versione minima di TLS

Per aumentare la sicurezza durante la comunicazione con AWS i servizi, configurali AWS SDK per JavaScript per utilizzare TLS 1.2 o versioni successive.

Transport Layer Security (TLS) è un protocollo utilizzato dai browser Web e da altre applicazioni per garantire la privacy e l'integrità dei dati scambiati su una rete.

> **Important**
>
> A partire dal 10 giugno 2024, abbiamo annunciato che TLS 1.3 è disponibile sugli endpoint delle API di AWS servizio in ciascuna delle regioni. AWS La versione AWS SDK per JavaScript v2 non negozia la versione TLS stessa. Utilizza invece la versione TLS determinata da Node.js, che è configurabile tramite. `https.Agent` AWS consiglia di utilizzare l'attuale versione Active LTS di Node.js.

Verificare e applicare TLS in Node.js

Quando si utilizza il AWS SDK per JavaScript con Node.js, il livello di sicurezza Node.js sottostante viene utilizzato per impostare la versione TLS.

Node.js 12.0.0 e versioni successive utilizzano una versione minima di OpenSSL 1.1.1b, che supporta TLS 1.3. Per impostazione predefinita, la AWS SDK per JavaScript v2 utilizza TLS 1.3 quando disponibile, ma utilizza per impostazione predefinita una versione precedente se necessario.

Verificare la versione di OpenSSL e TLS

Per ottenere la versione di OpenSSL utilizzata da Node.js sul computer, eseguire il seguente comando.

```
node -p process.versions
```

La versione di OpenSSL nell'elenco è la versione utilizzata da Node.js, come mostrato nell'esempio seguente.

```
openssl: '1.1.1b'
```

Per ottenere la versione di TLS utilizzata da Node.js sul computer, avviare la shell Node ed eseguire i seguenti comandi, in ordine.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

L'ultimo comando restituisce la versione TLS, come mostrato nell'esempio seguente.

```
'TLSv1.3'
```

Node.js utilizza per impostazione predefinita questa versione di TLS e tenta di negoziare un'altra versione di TLS se una chiamata non ha esito positivo.

Verifica delle versioni TLS minime e massime supportate

Gli sviluppatori possono verificare le versioni TLS minime e massime supportate in Node.js utilizzando lo script seguente:

```
var tls = require("tls");
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +
 tls.DEFAULT_MAX_VERSION);
```

L'ultimo comando restituisce la versione TLS minima e massima predefinita, come illustrato nell'esempio seguente.

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

Applicazione di una versione minima di TLS

Node.js negozia una versione di TLS quando una chiamata non riesce. È possibile applicare la versione TLS minima consentita durante questa negoziazione, sia quando si esegue uno script dalla riga di comando che per richiesta nel codice. JavaScript

Per specificare la versione TLS minima dalla riga di comando, è necessario utilizzare Node.js versione 11.4.0 o successiva. Per installare una versione specifica di Node.js, installare prima Node Version Manager (nvm) utilizzando la procedura indicata nell'argomento relativo all'installazione e all'aggiornamento di Node Version Manager. Quindi eseguire i seguenti comandi per installare e utilizzare una versione specifica di Node.js.

```
nvm install 11
nvm use 11
```

Enforcing TLS 1.2

Per stabilire che TLS 1.2 sia la versione minima consentita, specificare l'argomento `--tls-min-v1.2` durante l'esecuzione dello script, come mostrato nell'esempio seguente.

```
node --tls-min-v1.2 yourScript.js
```

Per specificare la versione TLS minima consentita per una richiesta specifica nel JavaScript codice, utilizzate il `httpOptions` parametro per specificare il protocollo, come illustrato nell'esempio seguente.

```
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

Per far sì che TLS 1.3 sia la versione minima consentita, specificate l'`--tls-min-v1.3`argomento durante l'esecuzione dello script, come illustrato nell'esempio seguente.

```
node --tls-min-v1.3 yourScript.js
```

Per specificare la versione TLS minima consentita per una richiesta specifica nel JavaScript codice, utilizzate il `httpOptions` parametro per specificare il protocollo, come illustrato nell'esempio seguente.

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

Verificare e applicare TLS in uno script del browser

Quando utilizzate l'SDK per JavaScript in uno script del browser, le impostazioni del browser controllano la versione di TLS utilizzata. La versione di TLS utilizzata dal browser non può essere individuata o impostata dallo script e deve essere configurata dall'utente. Per verificare e applicare la versione di TLS utilizzata in uno script del browser, consultare le istruzioni relative al browser specifico.

Microsoft Internet Explorer

1. Apri Internet Explorer.
2. Dalla barra dei menu, scegli Strumenti - Opzioni Internet - scheda Avanzate.
3. Scorri verso il basso fino alla categoria Sicurezza, seleziona manualmente la casella di opzione Usa TLS 1.2.
4. Fai clic su OK.
5. Chiudi il browser e riavvia Internet Explorer.

Microsoft Edge

1. Nella casella di ricerca del menu Windows, digita*Internet options*.
2. In Best match, fai clic su Opzioni Internet.
3. Nella finestra Proprietà Internet, nella scheda Avanzate, scorri verso il basso fino alla sezione Sicurezza.
4. Seleziona la casella di controllo User TLS 1.2.
5. Fai clic su OK.

Google Chrome

1. Apri Google Chrome.
2. Fai clic su Alt F e seleziona Impostazioni.
3. Scorri verso il basso e seleziona Mostra impostazioni avanzate... .
4. Scorri verso il basso fino alla sezione Sistema e fai clic su Apri impostazioni proxy... .
5. Seleziona la scheda Avanzate.
6. Scorri verso il basso fino alla categoria Sicurezza, seleziona manualmente la casella di opzione Usa TLS 1.2.
7. Fai clic su OK.
8. Chiudi il browser e riavvia Google Chrome.

Mozilla Firefox

1. Apri Firefox.
2. Nella barra degli indirizzi, digita about:config e premi Invio.
3. Nel campo Cerca, inserisci tls. Trova e fai doppio clic sulla voce security.tls.version.min.
4. Imposta il valore intero su 3 per forzare il protocollo TLS 1.2 a diventare quello predefinito.
5. Fai clic su OK.
6. Chiudi il browser e riavvia Mozilla Firefox.

Apple Safari

Non ci sono opzioni per abilitare i protocolli SSL. Se utilizzi la versione 7 o successiva di Safari, TLS 1.2 viene abilitato automaticamente.

Risorse aggiuntive

I seguenti link forniscono risorse aggiuntive da utilizzare con l'AWS SDK per JavaScript.

AWS SDKs e guida di riferimento agli strumenti

La AWS SDKs and Tools Reference Guide contiene anche impostazioni, funzionalità e altri concetti fondamentali comuni a molti di AWS SDKs.

JavaScript Forum SDK

Puoi trovare domande e discussioni su argomenti di interesse per gli utenti dell'SDK JavaScript nel Forum JavaScript SDK.

JavaScript SDK e guida per sviluppatori su GitHub

Sono disponibili diversi repository GitHub per l'SDK. JavaScript

- L'attuale SDK per JavaScript è disponibile nel repository SDK.
- L'SDK for JavaScript Developer Guide (questo documento) è disponibile in formato markdown nel proprio repository di documentazione.
- Alcuni estratti del codice di esempio incluso in questa guida sono disponibili nell'archivio dei codici di esempio per SDK.

JavaScript SDK su Gitter

Puoi anche trovare domande e discussioni sull'SDK for JavaScript nella community SDK su JavaScript Gitter.

Cronologia dei documenti per AWS SDK per JavaScript

- Versione SDK: consulta JavaScript Riferimento API
- Ultimo aggiornamento importante della documentazione: 31 marzo 2022

Cronologia dei documenti

La tabella seguente descrive le modifiche importanti apportate in ogni versione successiva AWS SDK per JavaScript a maggio 2018. Per ricevere notifiche sugli aggiornamenti di questa documentazione, è possibile sottoscrivere un feed RSS.

Modifica	Descrizione	Data
TLS 1.3 è ora supportato su tutti gli endpoint delle API AWS di servizio in tutte le regioni	Versione TLS e metodo aggiornati per la registrazione della versione TLS.	10 aprile 2025
Applicazione di una versione minima di TLS	Sono state aggiunte informazioni su TLS 1.3.	31 marzo 2022
Visualizzazione di foto in un bucket Amazon S3 da un browser	Aggiunto un esempio per la visualizzazione in modo semplice delle foto in album di foto esistenti.	13 maggio 2019
Impostazione delle credenziali in Node.js, nuove scelte per il caricamento delle credenziali	Aggiunte informazioni sulle credenziali che vengono caricate dal provider di credenziali ECS o da un processo di credenziali configurato.	25 aprile 2019
Credenziali utilizzando un processo di credenziali configurato	Aggiunte informazioni sulle credenziali che vengono	25 aprile 2019

	caricate da un processo di credenziali configurato.	
Novità Guida introduttiva a uno script del browser	Lo script Getting Started in a Browser è stato riscritto per semplificare l'esempio e per accedere al servizio Amazon Polly per inviare testo e restituire sintesi vocale riproducibile nel browser. Consulta Nozioni di base su uno script di browser per i nuovi contenuti.	14 luglio 2018
Nuovi esempi di codice Amazon SNS	Sono stati aggiunti quattro nuovi esempi di codice Node.js per lavorare con Amazon SNS. Vedi Amazon SNS Examples per il codice di esempio.	29 giugno 2018
Nuova guida introduttiva in Node.js	Le Nozioni di base su Node.js sono state riscritte per comprendere l'utilizzo di codici di esempio aggiornati e per fornire maggiori dettagli su come creare il file `package.json` e lo stesso codice Node.js. Consulta Nozioni di base su Node.js per i nuovi contenuti.	4 giugno 2018

Aggiornamenti precedenti

La tabella seguente descrive le modifiche importanti apportate in ogni versione AWS SDK per JavaScript precedente a giugno 2018.

Modifica	Descrizione	Data
Nuovi esempi di AWS Elemental MediaConvert codice	Sono stati aggiunti tre nuovi esempi di codice Node.js con AWS Elemental MediaConvert cui lavorare. Per il codice di esempio, consulta AWS Elemental MediaConvert Esempi.	21 maggio 2018
Nuova modifica sul GitHub pulsante	L'intestazione di ogni argomento ora include un pulsante che porta alla versione markdown dello stesso argomento, in GitHub modo da poter apportare modifiche per migliorare l'accuratezza e la completezza della guida.	21 febbraio 2018
Nuovo argomento su endpoint personalizzati	Sono state aggiunte informazioni sul formato e sull'utilizzo degli endpoint personalizzati per effettuare le chiamate API. Consultare Specificazione degli endpoint personalizzati.	20 febbraio 2018
Guida per sviluppatori SDK for Developer su JavaScript GitHub	L'SDK for JavaScript Developer Guide è disponibile in formato markdown nel proprio repository di documentazione. È possibile pubblicare i problemi che si desidera gestire con la guida o sottoporre le richieste pull per inviare le modifiche proposte.	16 febbraio 2018

Modifica	Descrizione	Data
Nuovo esempio di codice Amazon DynamoDB	È stato aggiunto un nuovo esempio di codice Node.js per l'aggiornamento di una tabella DynamoDB utilizzando il Document Client. Per il codice di esempio, consulta Utilizzo del DynamoDB Document Client.	14 febbraio 2018
Nuovo argomento sull'accesso a SDK	È stato aggiunto un argomento che descrive come registrare le chiamate API effettuate con l'SDK for, incluse informazioni sull'utilizzo di JavaScript un logger di terze parti. Consultar e Registrazione delle chiamate AWS SDK per JavaScript.	5 febbraio 2018
Aggiornamento argomento sull'impostazione di una regione	L'argomento che descrive come impostare la regione usata con l'SDK è stato aggiornato e ampliato, tra cui le informazioni sull'ordine di precedenza per impostare la regione. Consultare Impostazi one della regione AWS.	12 dicembre 2017
Nuovi esempi di codice Amazon SES	La sezione con esempi di codice SDK è stata aggiornat a per includere cinque nuovi esempi di utilizzo di Amazon SES. Per ulteriori informazioni su questi codici di esempio, consulta Esempi di servizi Amazon Simple Email.	9 novembre 2017

Modifica	Descrizione	Data
Miglioramenti della fruibilità	Sulla base di test recenti, sono state effettuate una serie di modifiche per migliorare la fruibilità della documenta zione. • Gli esempi di codice sono più chiaramente identificati come mirati per l'esecuzione su browser o Node.js. • I link alla tabella dei contenuti non saltano più immediatamente ad altri contenuti Web, tra cui la documentazione di riferimen to dell'API. • Include ulteriori collegame nti nella sezione Guida introduttiva ai dettagli sull'ottenimento AWS delle credenziali. • Fornisce ulteriori informazi oni sulle caratteristiche Node.js comuni necessari e per utilizzare il kit SDK. Per ulteriori informazioni, consulta Considerazioni su Node.js.	9 agosto 2017

Modifica	Descrizione	Data
Nuovi esempi di codice DynamoDB	La sezione con esempi di codice SDK è stata aggiornat a per riscrivere i due esempi precedenti e aggiungere tre nuovi esempi per lavorare con DynamoDB. Per ulteriori informazioni su questi codici di esempio, consulta Esempi di Amazon DynamoDB.	21 giugno 2017
Nuovi esempi di codice IAM	La sezione con esempi di codice SDK è stata aggiornat a per includere cinque nuovi esempi di utilizzo di IAM. Per ulteriori informazioni su questi codici di esempio, consulta AWS Esempi IAM.	23 dicembre 2016
CloudWatch Nuovi esempi di codice e Amazon SQS	La sezione con esempi di codice SDK è stata aggiornata per includere nuovi esempi per lavorare con CloudWatch e con Amazon SQS. Per ulteriori informazioni su questi codici di esempio, consulta CloudWatc h Esempi Amazon e Esempi di Amazon SQS.	20 dicembre 2016

Modifica	Descrizione	Data
Nuovi esempi di EC2 codice Amazon	La sezione con esempi di codice SDK è stata aggiornata per includere cinque nuovi esempi per lavorare con Amazon EC2. Per ulteriori informazioni su questi codici di esempio, consulta EC2 Esempi Amazon.	15 dicembre 2016
Elenco dei browser supportati reso più visibile	All'elenco dei browser supportati dall'SDK for JavaScript, precedentemente disponibile nell'argomento Prerequisiti, è stato assegnato un argomento specifico per renderlo più visibile nel sommario.	16 novembre 2016
Pubblicazione iniziale della nuova Guida per gli sviluppatori	La precedente Guida per gli sviluppatori è ora considerata obsoleta. La nuova Guida per gli sviluppatori è stata riorganizzata per rendere più semplice l'individuazione delle informazioni. Quando JavaScript gli scenari di Node.js o del browser presentano considerazioni speciali, queste vengono identificate come appropriate. La guida fornisce inoltre esempi di codice aggiuntivi che sono meglio organizzati in modo da renderli più semplici e rapidi da trovare.	28 Ottobre 2016