Panduan Pengembang untuk SDK v2

AWS SDK untuk JavaScript

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS SDK untuk JavaScript: Panduan Pengembang untuk SDK v2

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

..... ix
Apa itu AWS SDK untuk JavaScript? 1
Pemeliharaan dan dukungan untuk versi utama SDK 1
Menggunakan SDK dengan Node.js 2
Menggunakan SDK dengan AWS Amplify 2
Menggunakan SDK dengan Browser Web 2
Kasus Penggunaan Umum 2
Tentang Contoh 3
Memulai 4
Memulai Skrip Browser 4
Skenario 4
Langkah 1: Buat Kolam Identitas Amazon Cognito 5
Langkah 2: Tambahkan Kebijakan ke Peran IAM yang Dibuat 6
Langkah 3: Buat Halaman HTML 7
Langkah 4: Tulis Script Browser 8
Langkah 5: Jalankan Sampel 9
Sampel lengkap 10
Kemungkinan Peningkatan 11
Memulai di Node.js 12
Skenario 12
Tugas Prasyarat 12
Langkah 1: Instal SDK dan Dependensi 13
Langkah 2: Konfigurasikan Kredensial Anda 13
Langkah 3: Buat Package JSON untuk Proyek 14
Langkah 4: Tulis Kode Node.js 15
Langkah 5: Jalankan Sampel 16
Menyiapkan SDK untuk JavaScript 17
Prasyarat 17
Menyiapkan Lingkungan AWS Node.js 17
Web Browser Didukung 18
Menginstal SDK 19
Instalasi Menggunakan Bower 20
Memuat SDK 20
Upgrade Dari Versi 1 21

Konversi Otomatis Jenis Base64 dan Timestamp pada Input/Output 21
Dipindahkan response.data. RequestId untuk response.Requestid 22
Elemen Pembungkus Terpapar 23
Properti Klien yang Jatuh 28
Mengkonfigurasi SDK untuk JavaScript 29
Menggunakan Objek Konfigurasi Global 29
Pengaturan Konfigurasi Global 30
Pengaturan Konfigurasi Per Layanan 32
Data Konfigurasi yang Tidak Dapat Diubah 32
Mengatur AWS Wilayah 33
Dalam Konstruktor Kelas Klien 33
Menggunakan Objek Konfigurasi Global 33
Menggunakan Variabel Lingkungan 33
Menggunakan File Config Bersama 33
Urutan Prioritas untuk Mengatur Wilayah 34
Menentukan Titik Akhir Kustom 35
Format String Titik Akhir 35
Titik akhir untuk Wilayah ap-northeast-3 35
Endpoint untuk MediaConvert 35
Otentikasi SDK dengan AWS 36
Memulai sesi portal AWS akses 37
Informasi otentikasi lebih lanjut 38
Mengatur Kredensial 38
Praktik Terbaik untuk Kredensial 39
Mengatur Kredensial di Node.js 40
Menyetel Kredensial di Browser Web 45
Mengunci Versi API 55
Mendapatkan Versi API 55
Pertimbangan Node.js 55
Menggunakan Modul Node.js Built-In 56
Menggunakan Paket NPM 56
Mengkonfigurasi MaxSockets di Node.js 57
Menggunakan Kembali Koneksi dengan Keep-Alive di Node.js 58
Mengkonfigurasi Proxy untuk Node.js 59
Mendaftarkan Bundel Sertifikat di Node.js 60
Pertimbangan Skrip Browser 60

Membangun SDK untuk Browser 61
Berbagi Sumber Daya Lintas Orisinil (CORS) 64
Bundling dengan Webpack 68
Menginstal Webpack 68
Mengkonfigurasi Webpack 69
Menjalankan Webpack 70
Menggunakan Webpack Bundle 71
Mengimpor Layanan Individu 71
Bundling untuk Node.js 72
Bekerja dengan Layanan 74
Membuat dan Memanggil Objek Layanan 75
Membutuhkan Layanan Individu 76
Membuat Objek Layanan 77
Mengunci Versi API dari Objek Layanan 78
Menentukan Parameter Objek Layanan 78
AWS SDK untuk JavaScript Panggilan Pencatatan 79
Menggunakan Logger Pihak Ketiga 79
Layanan Panggilan Secara Asinkron 80
Mengelola Panggilan Asinkron 80
Menggunakan Fungsi Callback 81
Menggunakan Request Object Event Listener 83
Menggunakan async/await 88
Menggunakan Janji 89
Menggunakan Response Object 91
Mengakses Data yang Dikembalikan di Objek Response 92
Paging Melalui Data yang Dikembalikan 93
Mengakses Informasi Kesalahan dari Objek Respons 93
Mengakses Objek Permintaan Asal 94
Bekerja dengan JSON 94
JSON sebagai Parameter Objek Layanan 95
Mengembalikan Data sebagai JSON 96
Percobaan ulang 96
Perilaku coba lagi berbasis backoff eksponensial 97
SDK untuk Contoh JavaScript Kode 100
CloudWatch Contoh Amazon 100
Membuat Alarm di Amazon CloudWatch 101

Menggunakan Alarm Actions di Amazon CloudWatch 105
Mendapatkan Metrik dari Amazon CloudWatch 109
Mengirim Acara ke CloudWatch Acara Amazon 112
Menggunakan Filter Berlangganan di CloudWatch Log Amazon 118
Contoh Amazon DynamoDB 122
Membuat dan Menggunakan Tabel di DynamoDB 123
Membaca dan Menulis Satu Item di DynamoDB 128
Membaca dan Menulis Item dalam Batch di DynamoDB 132
Meminta dan Memindai Tabel DynamoDB 135
Menggunakan Klien Dokumen DynamoDB 139
EC2 Contoh Amazon 145
Membuat EC2 Instans Amazon 146
Mengelola EC2 Instans Amazon 148
Bekerja dengan Amazon EC2 Key Pairs 154
Menggunakan Wilayah dan Availability Zone dengan Amazon EC2 158
Bekerja dengan Grup Keamanan di Amazon EC2 160
Menggunakan Alamat IP Elastis di Amazon EC2 164
MediaConvert Contoh 169
Mendapatkan Endpoint Khusus Wilayah Anda 169
Menciptakan dan Mengelola Pekerjaan 171
Menggunakan Job Template 179
AWS Contoh IAM 187
Mengelola Pengguna IAM 188
Bekerja dengan Kebijakan IAM 193
Mengelola Kunci Akses IAM 199
Bekerja dengan Sertifikat Server IAM 204
Mengelola Alias Akun IAM 209
Contoh Kinesis Amazon 212
Menangkap Kemajuan Gulir Halaman Web dengan Amazon Kinesis 213
Contoh-contoh Amazon S3 219
Contoh Browser Amazon S3 220
Amazon S3 Node.js Contoh 249
Contoh Amazon SES 270
Mengelola Identitas 271
Bekerja dengan Template Email 276
Mengirim Email Menggunakan Amazon SES 282

Menggunakan Filter Alamat IP 288
Menggunakan Aturan Tanda Terima 293
Contoh Amazon SNS 298
Mengelola Topik 299
Menerbitkan Pesan ke Topik 305
Mengelola Langganan 307
Mengirim Pesan SMS 313
Amazon SQS Contoh 319
Menggunakan Antrian di Amazon SQS 320
Mengirim dan Menerima Pesan di Amazon SQS 325
Mengelola Batas Waktu Visibilitas di Amazon SQS 328
Mengaktifkan Polling Panjang di Amazon SQS 331
Menggunakan Antrian Surat Mati di Amazon SQS 335
Tutorial 338
Tutorial: Menyiapkan Node.js pada EC2 Instans Amazon 338
Prasyarat 338
Prosedur 338
Membuat Gambar Mesin Amazon 340
Sumber Daya Terkait 340
Referensi API dan Changelog 341
SDK Changelog aktif GitHub 341
Migrasi ke v3 342
Keamanan 343
Perlindungan data 343
Identity and Access Management 345
Audiens 345
Mengautentikasi dengan identitas 346
Mengelola akses menggunakan kebijakan 349
Bagaimana Layanan AWS bekerja dengan IAM 352
Memecahkan masalah AWS identitas dan akses 352
Validasi Kepatuhan 354
Ketahanan 356
Keamanan Infrastruktur 356
Menegakkan versi minimum TLS 357
Verifikasi dan terapkan TLS di Node.js 357
Verifikasi dan terapkan TLS dalam skrip browser 360

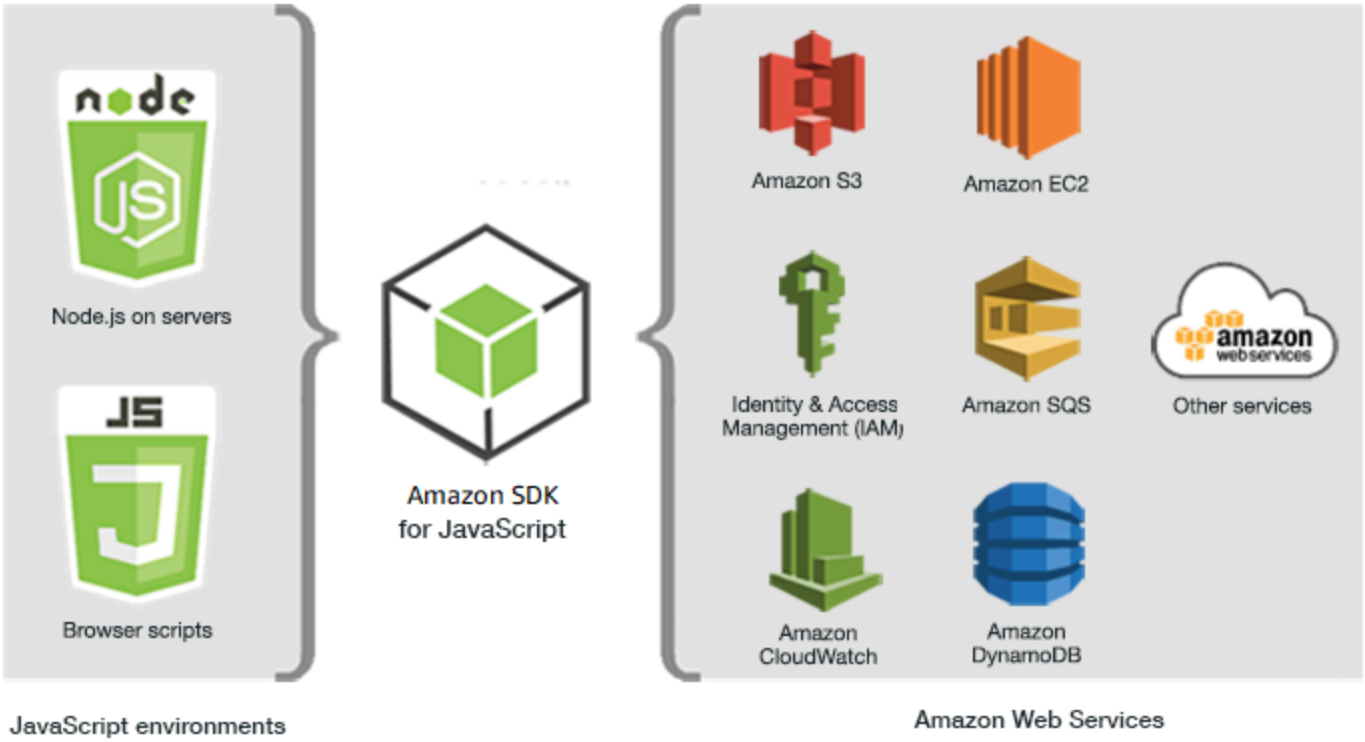
Sumber Daya Tambahan 363
AWS SDKs dan Panduan Referensi Alat 363
JavaScript Forum SDK 363
JavaScript SDK dan Panduan Pengembang di GitHub 363
JavaScript SDK di Gitter 363
Riwayat Dokumen 364
Riwayat Dokumen 364
Pembaruan Sebelumnya 365

Kami mengumumkan yang akan datang end-of-support untuk AWS SDK untuk JavaScript v2. Kami menyarankan Anda bermigrasi ke AWS SDK untuk JavaScript v3. Untuk tanggal, detail tambahan, dan informasi tentang cara bermigrasi, silakan merujuk ke pengumuman tertaut.

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.

Apa itu AWS SDK untuk JavaScript?

AWS SDK untuk JavaScriptMenyediakan JavaScript API untuk AWS layanan. Anda dapat menggunakan JavaScript API untuk membangun pustaka atau aplikasi untuk Node.js atau browser.



Tidak semua layanan segera tersedia di SDK. Untuk mengetahui layanan mana yang saat ini didukung oleh AWS SDK untuk JavaScript, lihat https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md. Untuk informasi tentang SDK for JavaScript on GitHub, lihatSumber Daya Tambahan.

Pemeliharaan dan dukungan untuk versi utama SDK

Untuk informasi tentang pemeliharaan dan dukungan untuk versi utama SDK dan dependensi yang mendasarinya, lihat berikut ini di Panduan Referensi Alat AWS SDKs dan Alat berikut:

- AWS SDKs dan kebijakan pemeliharaan alat
- AWS SDKs dan matriks dukungan versi alat

Menggunakan SDK dengan Node.js

Node.js adalah runtime lintas platform untuk menjalankan aplikasi sisi server JavaScript . Anda dapat mengatur Node.js pada EC2 instance Amazon untuk dijalankan di server. Anda juga dapat menggunakan Node.js untuk menulis AWS Lambda fungsi sesuai permintaan.

Menggunakan SDK untuk Node.js berbeda dari cara Anda menggunakannya JavaScript di browser web. Perbedaannya berasal dari cara Anda memuat SDK dan bagaimana Anda mendapatkan kredensil yang diperlukan untuk mengakses layanan web tertentu. Ketika penggunaan tertentu APIs berbeda antara Node.js dan browser, perbedaan tersebut akan dipanggil keluar.

Menggunakan SDK dengan AWS Amplify

Untuk aplikasi web, seluler, dan hybrid berbasis browser, Anda juga dapat menggunakan AWS Amplify Library on GitHub, yang memperluas SDK untuk JavaScript, menyediakan antarmuka deklaratif.

> **Note**
>
> Kerangka kerja seperti AWS Amplify mungkin tidak menawarkan dukungan browser yang sama dengan SDK untuk. JavaScript Periksa dokumentasi kerangka kerja untuk detailnya.

Menggunakan SDK dengan Browser Web

Semua browser web utama mendukung eksekusi JavaScript. JavaScriptKode yang berjalan di browser web sering disebut client-side JavaScript.

Menggunakan SDK for JavaScript di browser web berbeda dari cara Anda menggunakannya untuk Node.js. Perbedaannya berasal dari cara Anda memuat SDK dan bagaimana Anda mendapatkan kredensil yang diperlukan untuk mengakses layanan web tertentu. Ketika penggunaan tertentu APIs berbeda antara Node.js dan browser, perbedaan tersebut akan dipanggil keluar.

Untuk daftar browser yang didukung oleh AWS SDK untuk JavaScript, lihatWeb Browser Didukung.

Kasus Penggunaan Umum

Menggunakan SDK untuk JavaScript skrip browser memungkinkan untuk mewujudkan sejumlah kasus penggunaan yang menarik. Berikut adalah beberapa ide untuk hal-hal yang dapat Anda

bangun dalam aplikasi browser dengan menggunakan SDK JavaScript untuk mengakses berbagai layanan web.

- Buat konsol khusus untuk AWS layanan tempat Anda mengakses dan menggabungkan fitur di seluruh Wilayah dan layanan untuk memenuhi kebutuhan organisasi atau proyek Anda dengan sebaik-baiknya.
- Gunakan Identitas Amazon Cognito untuk mengaktifkan akses pengguna yang diautentikasi ke aplikasi dan situs web browser Anda, termasuk penggunaan otentikasi pihak ketiga dari Facebook dan lainnya.
- Gunakan Amazon Kinesis untuk memproses aliran klik atau data pemasaran lainnya secara real time.
- Gunakan Amazon DynamoDB untuk persistensi data tanpa server seperti preferensi pengguna individu untuk pengunjung situs web atau pengguna aplikasi.
- Gunakan AWS Lambda untuk merangkum logika kepemilikan yang dapat Anda panggil dari skrip browser tanpa mengunduh dan mengungkapkan kekayaan intelektual Anda kepada pengguna.

Tentang Contoh

Anda dapat menelusuri SDK untuk JavaScript contoh di AWS Code Example Library.

Memulai dengan AWS SDK untuk JavaScript

AWS SDK untuk JavaScript Ini menyediakan akses ke layanan web baik dalam skrip browser atau Node.js. Bagian ini memiliki dua latihan memulai yang menunjukkan cara bekerja dengan SDK JavaScript di setiap JavaScript lingkungan ini.

Topik

- Memulai Skrip Browser
- Memulai di Node.js

Memulai Skrip Browser

Contoh skrip browser ini menunjukkan kepada Anda:

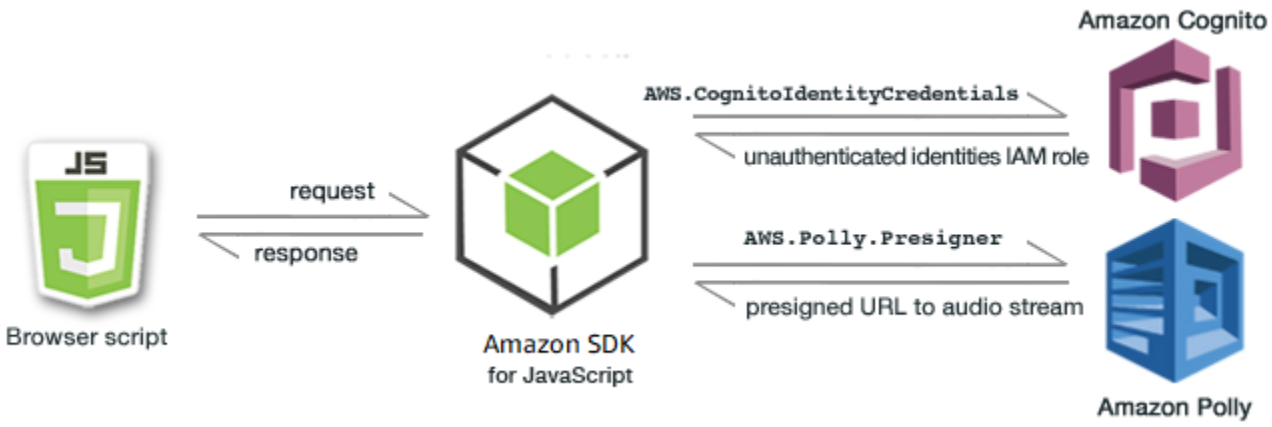
- Cara mengakses AWS layanan dari skrip browser menggunakan Amazon Cognito Identity.
- Cara mengubah teks menjadi ucapan yang disintesis menggunakan Amazon Polly.
- Cara menggunakan objek presigner untuk membuat URL presigned.

Skenario

Amazon Polly adalah layanan cloud yang mengubah teks menjadi ucapan yang hidup. Anda dapat menggunakan Amazon Polly untuk mengembangkan aplikasi yang meningkatkan keterlibatan dan aksesibilitas. Amazon Polly mendukung berbagai bahasa dan mencakup berbagai suara yang hidup. Untuk informasi selengkapnya tentang Amazon Polly, lihat Panduan Pengembang Amazon Polly.

Contoh menunjukkan cara mengatur dan menjalankan skrip browser sederhana yang mengambil teks yang Anda masukkan, mengirim teks itu ke Amazon Polly, dan kemudian mengembalikan URL audio teks yang disintesis untuk Anda mainkan. Skrip browser menggunakan Identitas Amazon Cognito untuk memberikan kredensyal yang diperlukan untuk mengakses layanan. AWS Anda akan melihat pola dasar untuk memuat dan menggunakan SDK untuk JavaScript skrip browser.

> **Note**
>
> Pemutaran pidato yang disintesis dalam contoh ini tergantung pada berjalan di browser yang mendukung audio HTML 5.



Skrip browser menggunakan SDK for JavaScript untuk mensintesis teks dengan menggunakan ini: APIs

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

Langkah 1: Buat Kolam Identitas Amazon Cognito

Dalam latihan ini, Anda membuat dan menggunakan kumpulan identitas Amazon Cognito untuk menyediakan akses tidak terautentikasi ke skrip browser Anda untuk layanan Amazon Polly. Membuat kumpulan identitas juga menciptakan dua peran IAM, satu untuk mendukung pengguna yang diautentikasi oleh penyedia identitas dan yang lainnya untuk mendukung pengguna tamu yang tidak diautentikasi.

Dalam latihan ini, kami hanya akan bekerja dengan peran pengguna yang tidak diautentikasi untuk menjaga tugas tetap fokus. Anda dapat mengintegrasikan dukungan untuk penyedia identitas dan pengguna yang diautentikasi nanti. Untuk informasi selengkapnya tentang menambahkan kumpulan identitas Amazon Cognito, lihat Tutorial: Membuat kumpulan identitas di Panduan Pengembang Amazon Cognito.

Untuk membuat kumpulan identitas Amazon Cognito

1. Masuk ke AWS Management Console dan buka konsol Amazon Cognito di. https://console.aws.amazon.com/cognito/
2. Di panel navigasi kiri, pilih Identity pool.
3. Pilih Buat kumpulan identitas.
4. Di Konfigurasikan kepercayaan kumpulan identitas, pilih Akses tamu untuk otentikasi pengguna.
5. Di Konfigurasi izin, pilih Buat peran IAM baru dan masukkan nama (misalnya, getStartedRole) di nama peran IAM.
6. Di Konfigurasi properti, masukkan nama (misalnya, getStartedPool) di nama kumpulan Identitas.
7. Di Tinjau dan buat, konfirmasikan pilihan yang Anda buat untuk kumpulan identitas baru Anda. Pilih Edit untuk kembali ke wizard dan mengubah pengaturan apa pun. Setelah selesai, pilih Buat kumpulan identitas.
8. Perhatikan ID kumpulan Identitas dan Wilayah kumpulan identitas Amazon Cognito yang baru dibuat. Anda membutuhkan nilai-nilai ini untuk menggantikan *IDENTITY_POOL_ID* dan *REGION* masukLangkah 4: Tulis Script Browser.

Setelah membuat kumpulan identitas Amazon Cognito, Anda siap menambahkan izin untuk Amazon Polly yang diperlukan oleh skrip browser Anda.

Langkah 2: Tambahkan Kebijakan ke Peran IAM yang Dibuat

Untuk mengaktifkan akses skrip browser ke Amazon Polly untuk sintesis ucapan, gunakan peran IAM yang tidak diautentikasi yang dibuat untuk kumpulan identitas Amazon Cognito Anda. Ini mengharuskan Anda untuk menambahkan kebijakan IAM ke peran tersebut. Untuk informasi selengkapnya tentang memodifikasi peran IAM, lihat Memodifikasi kebijakan izin peran di Panduan Pengguna IAM.

Untuk menambahkan kebijakan Amazon Polly ke peran IAM yang terkait dengan pengguna yang tidak diautentikasi

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi sebelah kiri, pilih Peran.
3. Pilih nama peran yang ingin Anda ubah (misalnya, getStartedRole), lalu pilih tab Izin.
4. Pilih Tambahkan izin, lalu pilih Lampirkan kebijakan.

5. Di halaman Tambahkan izin untuk peran ini, temukan lalu pilih kotak centang untuk AmazonPollyReadOnly.

> **Note**
>
> Anda dapat menggunakan proses ini untuk mengaktifkan akses ke AWS layanan apa pun.

6. Pilih Tambahkan izin.

Setelah Anda membuat kumpulan identitas Amazon Cognito dan menambahkan izin untuk Amazon Polly ke peran IAM Anda untuk pengguna yang tidak diautentikasi, Anda siap untuk membuat halaman web dan skrip browser.

Langkah 3: Buat Halaman HTML

Aplikasi sampel terdiri dari satu halaman HTML yang berisi antarmuka pengguna dan skrip browser. Untuk memulai, buat dokumen HTML dan salin konten berikut ke dalamnya. Halaman ini mencakup bidang input dan tombol, `<audio>` elemen untuk memainkan pidato yang disintesis, dan `<p>` elemen untuk menampilkan pesan. (Perhatikan bahwa contoh lengkap ditampilkan di bagian bawah halaman ini.)

Untuk informasi lebih lanjut tentang `<audio>` elemen, lihat 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>
```

Simpan file HTML, beri nama`polly.html`. Setelah Anda membuat antarmuka pengguna untuk aplikasi, Anda siap untuk menambahkan kode skrip browser yang menjalankan aplikasi.

Langkah 4: Tulis Script Browser

Hal pertama yang harus dilakukan saat membuat skrip browser adalah memasukkan SDK untuk JavaScript dengan menambahkan `<script>` elemen setelah `<audio>` elemen di halaman. Untuk menemukan SDK_VERSION_NUMBER saat ini, lihat Referensi API untuk SDK untuk Panduan Referensi API. JavaScript AWS SDK untuk JavaScript

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

Kemudian tambahkan `<script type="text/javascript">` elemen baru setelah entri SDK. Anda akan menambahkan skrip browser ke elemen ini. Tetapkan AWS Region dan kredensialnya untuk SDK. Selanjutnya, buat fungsi bernama `speakText()` yang akan dipanggil sebagai event handler oleh tombol.

Untuk mensintesis ucapan dengan Amazon Polly, Anda harus menyediakan berbagai parameter termasuk format suara output, laju pengambilan sampel, ID suara yang akan digunakan, dan teks untuk diputar ulang. Saat Anda awalnya membuat parameter, atur `Text:` parameter ke string kosong; `Text:` parameter akan diatur ke nilai yang Anda ambil dari `<input>` elemen di halaman web. Ganti *`IDENTITY_POOL_ID`* dan *`REGION`* dalam kode berikut dengan nilai yang tercantum dalamLangkah 1: Buat Kolam Identitas 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 mengembalikan ucapan yang disintesis sebagai aliran audio. Cara termudah untuk memutar audio itu di browser adalah dengan meminta Amazon Polly membuat audio tersedia di URL yang telah ditentukan sebelumnya yang kemudian dapat Anda atur sebagai `src` atribut `<audio>` elemen di halaman web.

Buat objek `AWS.Polly` layanan baru. Kemudian buat `AWS.Polly.Presigner` objek yang akan Anda gunakan untuk membuat URL presigned dari mana audio ucapan yang disintesis dapat diambil. Anda harus meneruskan parameter ucapan yang Anda tentukan serta objek `AWS.Polly` layanan yang Anda buat ke `AWS.Polly.Presigner` konstruktor.

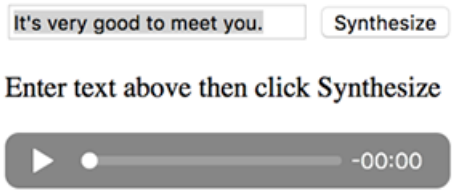
Setelah Anda membuat objek presigner, panggil `getSynthesizeSpeechUrl` metode objek itu, melewati parameter ucapan. Jika berhasil, metode ini mengembalikan URL pidato yang disintesis, yang kemudian Anda tetapkan ke `<audio>` elemen untuk pemutaran.

```
        // 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>
```

Langkah 5: Jalankan Sampel

Untuk menjalankan aplikasi sampel, muat `polly.html` ke browser web. Inilah yang harus menyerupai presentasi browser.



Masukkan frasa yang ingin diubah menjadi ucapan di kotak input, lalu pilih Sintesis. Saat audio siap diputar, sebuah pesan muncul. Gunakan kontrol pemutar audio untuk mendengar ucapan yang disintesis.

Sampel lengkap

Berikut adalah halaman HTML lengkap dengan skrip browser. Ini juga tersedia di sini 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>
```

Kemungkinan Peningkatan

Berikut adalah variasi pada aplikasi ini yang dapat Anda gunakan untuk mengeksplorasi lebih lanjut menggunakan SDK untuk JavaScript dalam skrip browser.

- Bereksperimenlah dengan format output suara lainnya.
- Tambahkan opsi untuk memilih salah satu dari berbagai suara yang disediakan oleh Amazon Polly.
- Integrasikan penyedia identitas seperti Facebook atau Amazon untuk digunakan dengan peran IAM yang diautentikasi.

Memulai di Node.js

Contoh kode Node.js ini menunjukkan:

- Cara membuat `package.json` manifes untuk proyek Anda.
- Cara menginstal dan menyertakan modul yang digunakan proyek Anda.
- Cara membuat objek layanan Amazon Simple Storage Service (Amazon S3) dari `AWS.S3` kelas klien.
- Cara membuat bucket Amazon S3 dan mengunggah objek ke bucket itu.

Skenario

Contoh menunjukkan cara mengatur dan menjalankan modul Node.js sederhana yang membuat bucket Amazon S3, lalu menambahkan objek teks ke dalamnya.

Karena nama bucket di Amazon S3 harus unik secara global, contoh ini menyertakan modul Node.js pihak ketiga yang menghasilkan nilai ID unik yang dapat Anda masukkan ke dalam nama bucket. Modul tambahan ini diberi nama`uuid`.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Buat direktori kerja untuk mengembangkan modul Node.js Anda. Beri nama direktori ini`awsnodesample`. Perhatikan bahwa direktori harus dibuat di lokasi yang dapat diperbarui oleh aplikasi. Misalnya, di Windows, jangan buat direktori di bawah "C:\Program Files”.
- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js. Anda dapat menemukan unduhan versi Node.js saat ini dan LTS untuk berbagai sistem operasi di https://nodejs. org/en/download/current/.

Daftar Isi

- Langkah 1: Instal SDK dan Dependensi
- Langkah 2: Konfigurasikan Kredensial Anda
- Langkah 3: Buat Package JSON untuk Proyek
- Langkah 4: Tulis Kode Node.js
- Langkah 5: Jalankan Sampel

Langkah 1: Instal SDK dan Dependensi

Anda menginstal SDK untuk JavaScript paket menggunakan npm (manajer paket Node.js).

Dari `awsnodesample` direktori dalam paket, ketik berikut ini di baris perintah.

```
npm install aws-sdk
```

Perintah ini menginstal SDK untuk JavaScript proyek Anda, dan memperbarui `package.json` untuk mencantumkan SDK sebagai dependensi proyek. Anda dapat menemukan informasi tentang paket ini dengan mencari “aws-sdk” di situs web npm.

Selanjutnya, instal `uuid` modul ke proyek dengan mengetikkan yang berikut ini di baris perintah, yang menginstal modul dan pembaruan`package.json`. Untuk informasi selengkapnya`uuid`, lihat halaman modul di https://www.npmjs.com/package/uuid.

```
npm install uuid
```

Paket-paket ini dan kode terkaitnya dipasang di `node_modules` subdirektori proyek Anda.

Untuk informasi selengkapnya tentang menginstal paket Node.js, lihat Mengunduh dan menginstal paket secara lokal dan Membuat Modul Node.js di situs web npm (Node.js package manager). Untuk informasi tentang mengunduh dan menginstal AWS SDK untuk JavaScript, lihatMenginstal SDK untuk JavaScript.

Langkah 2: Konfigurasikan Kredensial Anda

Anda perlu memberikan kredensyal AWS agar hanya akun Anda dan sumber dayanya yang diakses oleh SDK. Untuk informasi selengkapnya tentang mendapatkan kredensyal akun Anda, lihat. Otentikasi SDK dengan AWS

Untuk menyimpan informasi ini, kami sarankan Anda membuat file kredensyal bersama. Untuk mempelajari caranya, lihat Memuat Kredensial di Node.js dari File Kredensial Bersama. File kredensial Anda harus menyerupai contoh berikut.

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

Anda dapat menentukan apakah Anda telah menyetel kredensialnya dengan benar dengan mengeksekusi kode berikut dengan 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);
  }
});
```

Demikian pula, jika Anda telah mengatur wilayah Anda dengan benar di `config` file Anda, Anda dapat menampilkan nilai itu dengan menyetel variabel `AWS_SDK_LOAD_CONFIG` lingkungan ke nilai apa pun dan menggunakan kode berikut:

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

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

Langkah 3: Buat Package JSON untuk Proyek

Setelah Anda membuat direktori `awsnodesample` proyek, Anda membuat dan menambahkan `package.json` file untuk menyimpan metadata untuk proyek Node.js Anda. Untuk detail tentang penggunaan `package.json` dalam proyek Node.js, lihat Membuat file package.json.

Di direktori proyek, buat file baru bernama`package.json`. Kemudian tambahkan JSON ini ke 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"
}
```

Simpan file tersebut. Saat Anda menginstal modul yang Anda butuhkan, `dependencies` bagian file akan selesai. Anda dapat menemukan file JSON yang menunjukkan contoh dependensi ini di sini. GitHub

Langkah 4: Tulis Kode Node.js

Buat file baru bernama `sample.js` berisi kode contoh. Mulailah dengan menambahkan panggilan `require` fungsi untuk menyertakan SDK for JavaScript dan `uuid` modul sehingga tersedia untuk Anda gunakan.

Buat nama bucket unik yang digunakan untuk membuat bucket Amazon S3 dengan menambahkan nilai ID unik ke awalan yang dapat dikenali, dalam hal ini. `'node-sdk-sample-'` Anda menghasilkan ID unik dengan memanggil `uuid` modul. Kemudian buat nama untuk `Key` parameter yang digunakan untuk mengunggah objek ke bucket.

Buat `promise` objek untuk memanggil `createBucket` metode objek `AWS.S3` layanan. Pada respons yang berhasil, buat parameter yang diperlukan untuk mengunggah teks ke bucket yang baru dibuat. Menggunakan janji lain, panggil `putObject` metode untuk mengunggah objek teks ke ember.

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

Kode contoh ini dapat ditemukan di sini GitHub.

Langkah 5: Jalankan Sampel

Ketik perintah berikut untuk menjalankan sampel.

```
node sample.js
```

Jika unggahan berhasil, Anda akan melihat pesan konfirmasi di baris perintah. Anda juga dapat menemukan bucket dan objek teks yang diunggah di konsol Amazon S3.

Menyiapkan SDK untuk JavaScript

Topik di bagian ini menjelaskan cara menginstal SDK JavaScript untuk digunakan di browser web dan dengan Node.js. Ini juga menunjukkan cara memuat SDK sehingga Anda dapat mengakses layanan web yang didukung oleh SDK.

> **Note**
>
> Pengembang React Native harus menggunakan AWS Amplify untuk membuat proyek baru di AWS. Lihat aws-sdk-react-nativearsip untuk detailnya.

Topik

- Prasyarat
- Menginstal SDK untuk JavaScript
- Memuat SDK untuk JavaScript
- Memutakhirkan SDK untuk JavaScript dari Versi 1

Prasyarat

Sebelum Anda menggunakan AWS SDK untuk JavaScript, tentukan apakah kode Anda perlu dijalankan di Node.js atau browser web. Setelah itu, lakukan hal berikut:

- Untuk Node.js, instal Node.js di server Anda jika belum diinstal.
- Untuk browser web, identifikasi versi browser yang perlu Anda dukung.

Topik

- Menyiapkan Lingkungan AWS Node.js
- Web Browser Didukung

Menyiapkan Lingkungan AWS Node.js

Untuk menyiapkan lingkungan AWS Node.js di mana Anda dapat menjalankan aplikasi Anda, gunakan salah satu metode berikut:

- Pilih Amazon Machine Image (AMI) dengan Node.js yang sudah diinstal sebelumnya dan buat EC2 instance Amazon menggunakan AMI tersebut. Saat membuat EC2 instans Amazon Anda, pilih AMI Anda dari AWS Marketplace. AWS Marketplace Cari Node.js dan pilih opsi AMI yang menyertakan versi Node.js (32-bit atau 64-bit) yang sudah diinstal sebelumnya.
- Buat EC2 instance Amazon dan instal Node.js di atasnya. Untuk informasi selengkapnya tentang cara menginstal Node.js pada instans Amazon Linux, lihatTutorial: Menyiapkan Node.js pada EC2 Instans Amazon.
- Buat lingkungan tanpa server menggunakan AWS Lambda untuk menjalankan Node.js sebagai fungsi Lambda. Untuk informasi selengkapnya tentang penggunaan Node.js dalam fungsi Lambda, lihat Model Pemrograman (Node.js) di Panduan AWS Lambda Pengembang.
- Terapkan aplikasi Node.js Anda ke AWS Elastic Beanstalk. Untuk informasi selengkapnya tentang penggunaan Node.js dengan Elastic Beanstalk, lihat Menerapkan Aplikasi Node.js AWS Elastic Beanstalk ke dalam Panduan Pengembang.AWS Elastic Beanstalk

Web Browser Didukung

SDK untuk JavaScript mendukung semua browser web modern, termasuk versi minimum ini:

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

> **Note**
>
> Kerangka kerja seperti AWS Amplify mungkin tidak menawarkan dukungan browser yang sama dengan SDK untuk. JavaScript Periksa dokumentasi kerangka kerja untuk detailnya.

Menginstal SDK untuk JavaScript

Apakah dan bagaimana Anda menginstal AWS SDK untuk JavaScript tergantung apakah kode dijalankan dalam modul Node.js atau skrip browser.

Tidak semua layanan segera tersedia di SDK. Untuk mengetahui layanan mana yang saat ini didukung oleh AWS SDK untuk JavaScript, lihat https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md

Node

Cara yang lebih disukai untuk menginstal AWS SDK untuk JavaScript untuk Node.js adalah dengan menggunakan npm, manajer paket Node.js. Untuk melakukannya, ketik ini di baris perintah.

```
npm install aws-sdk
```

Jika Anda melihat pesan galat ini:

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

Ketik perintah ini di baris perintah:

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

Browser

Anda tidak perlu menginstal SDK untuk menggunakannya dalam skrip browser. Anda dapat memuat paket SDK yang dihosting langsung dari Amazon Web Services dengan skrip di halaman HTML Anda. Paket SDK yang dihosting mendukung subset AWS layanan yang menerapkan berbagi sumber daya lintas asal (CORS). Untuk informasi selengkapnya, lihat Memuat SDK untuk JavaScript.

Anda dapat membuat pembuatan kustom SDK tempat Anda memilih layanan web dan versi tertentu yang ingin Anda gunakan. Anda kemudian mengunduh paket SDK khusus Anda untuk pengembangan lokal dan menghostingnya untuk digunakan aplikasi Anda. Untuk informasi selengkapnya tentang membuat custom build SDK, lihatMembangun SDK untuk Browser.

Anda dapat mengunduh versi yang dapat didistribusikan yang diperkecil dan tidak diperkecil dari saat ini dari di: AWS SDK untuk JavaScript GitHub

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

Instalasi Menggunakan Bower

Bower adalah manajer paket untuk web. Setelah Anda menginstal Bower, Anda dapat menggunakannya untuk menginstal SDK. Untuk menginstal SDK menggunakan Bower, ketik berikut ini ke jendela terminal:

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

Memuat SDK untuk JavaScript

Cara Anda memuat SDK JavaScript tergantung pada apakah Anda memuatnya untuk dijalankan di browser web atau di Node.js.

Tidak semua layanan segera tersedia di SDK. Untuk mengetahui layanan mana yang saat ini didukung oleh AWS SDK untuk JavaScript, lihat https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md

Node.js

Setelah Anda menginstal SDK, Anda dapat memuat AWS paket dalam aplikasi node Anda menggunakan`require`.

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

React Native

Untuk menggunakan SDK dalam proyek React Native, instal SDK terlebih dahulu menggunakan npm:

```
npm install aws-sdk
```

Dalam aplikasi Anda, rujuk versi SDK yang kompatibel dengan React Native dengan kode berikut:

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

Browser

Cara tercepat untuk memulai SDK adalah memuat paket SDK yang dihosting langsung dari Amazon Web Services. Untuk melakukan ini, tambahkan `<script>` elemen ke halaman HTML Anda dalam bentuk berikut:

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

Untuk menemukan SDK_VERSION_NUMBER saat ini, lihat Referensi API untuk SDK untuk Panduan Referensi API. JavaScript AWS SDK untuk JavaScript

Setelah SDK dimuat di halaman Anda, SDK tersedia dari variabel global `AWS` (atau`window.AWS`).

Jika Anda menggabungkan kode dan dependensi modul menggunakan browserify, Anda memuat SDK menggunakan`require`, seperti yang Anda lakukan di Node.js.

Memutakhirkan SDK untuk JavaScript dari Versi 1

Catatan berikut membantu Anda meningkatkan SDK JavaScript dari versi 1 ke versi 2.

Konversi Otomatis Jenis Base64 dan Timestamp pada Input/Output

SDK sekarang secara otomatis mengkodekan dan mendekode nilai yang dikodekan base64, serta nilai stempel waktu, atas nama pengguna. Perubahan ini memengaruhi operasi apa pun di mana nilai base64 atau stempel waktu dikirim oleh permintaan atau dikembalikan dalam respons yang memungkinkan nilai yang dikodekan base64.

Kode pengguna yang sebelumnya dikonversi base64 tidak lagi diperlukan. Nilai yang dikodekan sebagai base64 sekarang dikembalikan sebagai objek buffer dari respons server dan juga dapat diteruskan sebagai input buffer. Misalnya, `SQS.sendMessage` parameter versi 1 berikut:

```
var params = {
```

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

Dapat ditulis ulang sebagai berikut.

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

Berikut adalah bagaimana pesan dibaca.

```
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"
});
```

Dipindahkan response.data. RequestId untuk response.Requestid

SDK sekarang menyimpan permintaan IDs untuk semua layanan di tempat yang konsisten pada `response` objek, bukan di dalam `response.data` properti. Ini meningkatkan konsistensi di seluruh layanan yang mengekspos permintaan dengan IDs cara yang berbeda. Ini juga merupakan perubahan besar yang mengganti nama `response.data.RequestId` properti menjadi `response.requestId` (`this.requestId`di dalam fungsi panggilan balik).

Dalam kode Anda, ubah yang berikut ini:

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

```
});
```

Ke hal berikut:

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

Elemen Pembungkus Terpapar

Jika Anda menggunakan`AWS.ElastiCache`,`AWS.RDS`, atau`AWS.Redshift`, Anda harus mengakses respons melalui properti output tingkat atas dalam respons untuk beberapa operasi.

Misalnya, `RDS.describeEngineDefaultParameters` metode yang digunakan untuk mengembalikan yang berikut ini.

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

Sekarang mengembalikan yang berikut ini.

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

Daftar operasi yang terpengaruh untuk setiap layanan ditunjukkan pada tabel berikut.

Kelas Klien	Operasi

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

Kelas Klien	Operasi
	deleteReplicationGroup describeEngineDefaultParameters modifyCacheCluster modifyCacheSubnetGroup modifyReplicationGroup purchaseReservedCacheNodesOffering rebootCacheCluster revokeCacheSecurityGroupIngress

Kelas Klien	Operasi
AWS.RDS	addSourceIdentifierToSubscription authorizeDBSecurityGroupIngress copyDBSnapshot createDBInstance createDBInstanceReadReplica createDBParameterGroup createDBSecurityGroup createDBSnapshot createDBSubnetGroup createEventSubscription createOptionGroup deleteDBInstance deleteDBSnapshot deleteEventSubscription describeEngineDefaultParameters modifyDBInstance modifyDBSubnetGroup modifyEventSubscription modifyOptionGroup promoteReadReplica purchaseReservedDBInstancesOffering

Kelas Klien	Operasi

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

Kelas Klien	Operasi
AWS.Redshift	authorizeClusterSecurityGroupIngress authorizeSnapshotAccess copyClusterSnapshot createCluster createClusterParameterGroup createClusterSecurityGroup createClusterSnapshot createClusterSubnetGroup createEventSubscription createHsmClientCertificate createHsmConfiguration deleteCluster deleteClusterSnapshot describeDefaultClusterParameters disableSnapshotCopy enableSnapshotCopy modifyCluster modifyClusterSubnetGroup modifyEventSubscription modifySnapshotCopyRetentionPeriod

Kelas Klien	Operasi

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

Properti Klien yang Jatuh

`.client`Properti `.Client` dan telah dihapus dari objek layanan. Jika Anda menggunakan `.Client` properti pada kelas layanan atau `.client` properti pada instance objek layanan, hapus properti ini dari kode Anda.

Kode berikut yang digunakan dengan SDK versi 1 untuk JavaScript:

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

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

Harus diubah ke kode berikut.

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

Mengkonfigurasi SDK untuk JavaScript

Sebelum Anda menggunakan SDK for JavaScript untuk memanggil layanan web menggunakan API, Anda harus mengonfigurasi SDK. Minimal, Anda harus mengonfigurasi pengaturan ini:

- Wilayah di mana Anda akan meminta layanan.
- Kredensil yang mengotorisasi akses Anda ke sumber daya SDK.

Selain pengaturan ini, Anda mungkin juga harus mengonfigurasi izin untuk AWS sumber daya Anda. Misalnya, Anda dapat membatasi akses ke bucket Amazon S3 atau membatasi tabel Amazon DynamoDB untuk akses hanya-baca.

Panduan Referensi AWS SDKs and Tools juga berisi pengaturan, fitur, dan konsep dasar lainnya yang umum di antara banyak. AWS SDKs

Topik di bagian ini menjelaskan berbagai cara untuk mengkonfigurasi SDK JavaScript untuk Node.js dan JavaScript berjalan di browser web.

Topik

- Menggunakan Objek Konfigurasi Global
- Mengatur AWS Wilayah
- Menentukan Titik Akhir Kustom
- Otentikasi SDK dengan AWS
- Mengatur Kredensial
- Mengunci Versi API
- Pertimbangan Node.js
- Pertimbangan Skrip Browser
- Bundling Aplikasi dengan Webpack

Menggunakan Objek Konfigurasi Global

Ada dua cara untuk mengkonfigurasi SDK:

- Atur konfigurasi global menggunakan`AWS.Config`.
- Teruskan informasi konfigurasi tambahan ke objek layanan.

Menyetel konfigurasi global dengan `AWS.Config` seringkali lebih mudah untuk memulai, tetapi konfigurasi tingkat layanan dapat memberikan kontrol lebih besar atas layanan individual. Konfigurasi global yang ditentukan oleh `AWS.Config` menyediakan pengaturan default untuk objek layanan yang Anda buat selanjutnya, menyederhanakan konfigurasi mereka. Namun, Anda dapat memperbarui konfigurasi objek layanan individual ketika kebutuhan Anda bervariasi dari konfigurasi global.

Pengaturan Konfigurasi Global

Setelah Anda memuat `aws-sdk` paket dalam kode Anda, Anda dapat menggunakan variabel `AWS` global untuk mengakses kelas SDK dan berinteraksi dengan layanan individual. SDK menyertakan objek konfigurasi global`AWS.Config`, yang dapat Anda gunakan untuk menentukan setelan konfigurasi SDK yang diperlukan oleh aplikasi Anda.

Konfigurasikan SDK dengan menyetel `AWS.Config` properti sesuai dengan kebutuhan aplikasi Anda. Tabel berikut merangkum `AWS.Config` properti yang biasa digunakan untuk mengatur konfigurasi SDK.

Opsi konfigurasi	Deskripsi
`credentials`	Diperlukan. Menentukan kredensil yang digunakan untuk menentukan akses ke layanan dan sumber daya.
`region`	Diperlukan. Menentukan Wilayah di mana permintaan untuk layanan dibuat.
`maxRetries`	Tidak wajib. Menentukan jumlah maksimum kali permintaan yang diberikan dicoba ulang.
`logger`	Tidak wajib. Menentukan objek logger yang informasi debugging ditulis.
`update`	Tidak wajib. Memperbarui konfigurasi saat ini dengan nilai baru.

Untuk informasi selengkapnya tentang objek konfigurasi, lihat `Class: AWS.Config`di Referensi API.

Contoh Konfigurasi Global

Anda harus mengatur Wilayah dan kredensialnya. `AWS.Config` Anda dapat mengatur properti ini sebagai bagian dari `AWS.Config` konstruktor, seperti yang ditunjukkan dalam contoh skrip browser berikut:

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

Anda juga dapat mengatur properti ini setelah membuat `AWS.Config` menggunakan `update` metode, seperti yang ditunjukkan dalam contoh berikut yang memperbarui Wilayah:

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

Anda bisa mendapatkan kredensi default Anda dengan memanggil `getCredentials` metode statis: `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);
  }
});
```

Demikian pula, jika Anda telah mengatur wilayah Anda dengan benar di `config` file Anda, Anda mendapatkan nilai itu dengan menyetel variabel `AWS_SDK_LOAD_CONFIG` lingkungan disetel ke nilai apa pun dan memanggil `region` properti statis dari`AWS.config`:

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

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

Pengaturan Konfigurasi Per Layanan

Setiap layanan yang Anda gunakan dalam SDK untuk JavaScript diakses melalui objek layanan yang merupakan bagian dari API untuk layanan tersebut. Misalnya, untuk mengakses layanan Amazon S3 Anda membuat objek layanan Amazon S3. Anda dapat menentukan pengaturan konfigurasi yang spesifik untuk layanan sebagai bagian dari konstruktor untuk objek layanan tersebut. Ketika Anda menetapkan nilai konfigurasi pada objek layanan, konstruktor mengambil semua nilai konfigurasi yang digunakan oleh`AWS.Config`, termasuk kredensil.

Misalnya, jika Anda perlu mengakses EC2 objek Amazon di beberapa Wilayah, buat objek EC2 layanan Amazon untuk setiap Wilayah, lalu atur konfigurasi Wilayah dari setiap objek layanan yang sesuai.

```
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'});
```

Anda juga dapat menyetel nilai konfigurasi khusus untuk layanan saat mengonfigurasi SDK dengan. `AWS.Config` Objek konfigurasi global mendukung banyak opsi konfigurasi khusus layanan. Untuk informasi selengkapnya tentang konfigurasi khusus layanan, lihat `Class: AWS.Config`di Referensi AWS SDK untuk JavaScript API.

Data Konfigurasi yang Tidak Dapat Diubah

Perubahan konfigurasi global berlaku untuk permintaan untuk semua objek layanan yang baru dibuat. Objek layanan yang baru dibuat dikonfigurasi dengan data konfigurasi global saat ini terlebih dahulu dan kemudian opsi konfigurasi lokal apa pun. Pembaruan yang Anda buat ke `AWS.config` objek global tidak berlaku untuk objek layanan yang dibuat sebelumnya.

Objek layanan yang ada harus diperbarui secara manual dengan data konfigurasi baru atau Anda harus membuat dan menggunakan objek layanan baru yang memiliki data konfigurasi baru. Contoh berikut membuat objek layanan Amazon S3 baru dengan data konfigurasi baru:

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

Mengatur AWS Wilayah

Wilayah adalah kumpulan AWS sumber daya bernama di wilayah geografis yang sama. Contoh dari sebuah Wilayah adalah`us-east-1`, yang merupakan Wilayah AS Timur (Virginia N.). Anda menentukan Wilayah saat mengonfigurasi SDK JavaScript agar SDK mengakses sumber daya di Wilayah tersebut. Beberapa layanan hanya tersedia di Wilayah tertentu.

SDK for JavaScript tidak memilih Region secara default. Namun, Anda dapat mengatur Wilayah menggunakan variabel lingkungan, `config` file bersama, atau objek konfigurasi global.

Dalam Konstruktor Kelas Klien

Ketika Anda membuat instance objek layanan, Anda dapat menentukan Region untuk sumber daya tersebut sebagai bagian dari konstruktor kelas klien, seperti yang ditunjukkan di sini.

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

Menggunakan Objek Konfigurasi Global

Untuk mengatur Wilayah dalam JavaScript kode Anda, perbarui objek konfigurasi `AWS.Config` global seperti yang ditunjukkan di sini.

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

Untuk informasi selengkapnya tentang Wilayah saat ini dan layanan yang tersedia di setiap Wilayah, lihat AWS Wilayah dan Titik Akhir di Referensi Umum AWS.

Menggunakan Variabel Lingkungan

Anda dapat mengatur Wilayah menggunakan variabel `AWS_REGION` lingkungan. Jika Anda mendefinisikan variabel ini, SDK untuk JavaScript membacanya dan menggunakannya.

Menggunakan File Config Bersama

Sama seperti file kredensial bersama yang memungkinkan Anda menyimpan kredensil untuk digunakan oleh SDK, Anda dapat menyimpan Region dan pengaturan konfigurasi lainnya dalam file bersama bernama `config` yang digunakan oleh. SDKs Jika variabel `AWS_SDK_LOAD_CONFIG`

lingkungan telah disetel ke nilai apa pun, SDK untuk JavaScript secara otomatis mencari `config` file saat dimuat. Di mana Anda menyimpan `config` file tergantung pada sistem operasi Anda:

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

Jika Anda belum memiliki `config` file bersama, Anda dapat membuatnya di direktori yang ditunjuk. Dalam contoh berikut, `config` file menetapkan Region dan format output.

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

Untuk informasi selengkapnya tentang penggunaan file konfigurasi dan kredensial bersama, lihat Memuat Kredensial di Node.js dari File Kredensial Bersama atau File Konfigurasi dan Kredensi di Panduan Pengguna.AWS Command Line Interface

Urutan Prioritas untuk Mengatur Wilayah

Urutan prioritas untuk pengaturan Wilayah adalah sebagai berikut:

- Jika Region diteruskan ke konstruktor kelas klien, Region itu digunakan. Jika tidak, maka...
- Jika Region diatur pada objek konfigurasi global, Region tersebut akan digunakan. Jika tidak, maka...
- Jika variabel `AWS_REGION` lingkungan adalah nilai yang benar, Wilayah itu digunakan. Jika tidak, maka...
- Jika variabel `AMAZON_REGION` lingkungan adalah nilai yang benar, Wilayah itu digunakan. Jika tidak, maka...
- Jika variabel `AWS_SDK_LOAD_CONFIG` lingkungan disetel ke nilai apa pun dan file kredensial bersama (`~/.aws/credentials`atau jalur yang ditunjukkan oleh`AWS_SHARED_CREDENTIALS_FILE`) berisi Wilayah untuk profil yang dikonfigurasi, Wilayah tersebut akan digunakan. Jika tidak, maka...
- Jika variabel `AWS_SDK_LOAD_CONFIG` lingkungan disetel ke nilai apa pun dan file konfigurasi (`~/.aws/config`atau jalur yang ditunjukkan oleh`AWS_CONFIG_FILE`) berisi Wilayah untuk profil yang dikonfigurasi, Wilayah tersebut akan digunakan.

Menentukan Titik Akhir Kustom

Panggilan ke metode API di SDK for JavaScript dilakukan ke titik akhir URIs layanan. Secara default, titik akhir ini dibangun dari Wilayah yang telah Anda konfigurasikan untuk kode Anda. Namun, ada situasi di mana Anda perlu menentukan titik akhir kustom untuk panggilan API Anda.

Format String Titik Akhir

Nilai titik akhir harus berupa string dalam format:

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

Titik akhir untuk Wilayah ap-northeast-3

`ap-northeast-3`Wilayah di Jepang tidak dikembalikan oleh APIs enumerasi Wilayah, seperti. `EC2.describeRegions` Untuk menentukan titik akhir untuk Wilayah ini, ikuti format yang dijelaskan sebelumnya. Jadi EC2 titik akhir Amazon untuk Wilayah ini adalah

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

Endpoint untuk MediaConvert

Anda perlu membuat titik akhir khusus untuk digunakan. MediaConvert Setiap akun pelanggan diberi titik akhirnya sendiri, yang harus Anda gunakan. Berikut adalah contoh cara menggunakan endpoint khusus dengan 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
};
```

Untuk mendapatkan titik akhir API akun Anda, lihat `MediaConvert.describeEndpoints`di Referensi API.

Pastikan Anda menentukan Region yang sama dalam kode Anda sebagai Region di URI endpoint kustom. Ketidakcocokan antara setelan Wilayah dan URI titik akhir kustom dapat menyebabkan panggilan API gagal.

Untuk informasi selengkapnya MediaConvert, lihat `AWS.MediaConvert`kelas di Referensi API atau Panduan AWS Elemental MediaConvert Pengguna.

Otentikasi SDK dengan AWS

Anda harus menetapkan bagaimana kode Anda mengautentikasi AWS saat mengembangkan dengan Layanan AWS. Anda dapat mengonfigurasi akses terprogram ke AWS sumber daya dengan cara yang berbeda tergantung pada lingkungan dan AWS akses yang tersedia untuk Anda.

Untuk memilih metode otentikasi dan mengonfigurasinya untuk SDK, lihat Autentikasi dan akses di Panduan Referensi Alat AWS SDKs dan Alat.

Kami menyarankan agar pengguna baru yang berkembang secara lokal dan tidak diberi metode otentikasi oleh majikan mereka harus disiapkan. AWS IAM Identity Center Metode ini termasuk menginstal AWS CLI untuk kemudahan konfigurasi dan untuk masuk secara teratur ke portal AWS akses. Jika Anda memilih metode ini, lingkungan Anda harus berisi elemen-elemen berikut setelah Anda menyelesaikan prosedur untuk autentikasi IAM Identity Center di AWS SDKs dan Panduan Referensi Alat:

- Itu AWS CLI, yang Anda gunakan untuk memulai sesi portal AWS akses sebelum Anda menjalankan aplikasi Anda.
- AWS`config`File bersama yang memiliki `[default]` profil dengan serangkaian nilai konfigurasi yang dapat direferensikan dari SDK. Untuk menemukan lokasi file ini, lihat Lokasi file bersama di Panduan Referensi Alat AWS SDKs dan.
- `config`File bersama menetapkan `region`pengaturan. Ini menetapkan default AWS Region yang digunakan SDK untuk AWS permintaan. Wilayah ini digunakan untuk permintaan layanan SDK yang tidak ditentukan dengan Wilayah yang akan digunakan.
- SDK menggunakan konfigurasi penyedia token SSO profil untuk memperoleh kredensional sebelum mengirim permintaan ke. AWS`sso_role_name`Nilai, yang merupakan peran IAM yang terhubung ke set izin Pusat Identitas IAM, memungkinkan akses ke yang Layanan AWS digunakan dalam aplikasi Anda.

`config`File contoh berikut menunjukkan profil default yang diatur dengan konfigurasi penyedia token SSO. `sso_session`Pengaturan profil mengacu pada `sso-session`bagian bernama. `sso-session`Bagian ini berisi pengaturan untuk memulai sesi portal AWS akses.

```
[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
```

SDK for JavaScript tidak memerlukan paket tambahan (seperti `SSO` dan`SSOOIDC`) untuk ditambahkan ke aplikasi Anda untuk menggunakan autentikasi IAM Identity Center.

Memulai sesi portal AWS akses

Sebelum menjalankan aplikasi yang mengakses Layanan AWS, Anda memerlukan sesi portal AWS akses aktif agar SDK menggunakan autentikasi IAM Identity Center untuk menyelesaikan kredensialnya. Bergantung pada panjang sesi yang dikonfigurasi, akses Anda pada akhirnya akan kedaluwarsa dan SDK akan mengalami kesalahan otentikasi. Untuk masuk ke portal AWS akses, jalankan perintah berikut di AWS CLI.

```
aws sso login
```

Jika Anda mengikuti panduan dan memiliki pengaturan profil default, Anda tidak perlu memanggil perintah dengan `--profile` opsi. Jika konfigurasi penyedia token SSO Anda menggunakan profil bernama, perintahnya adalah`aws sso login --profile named-profile`.

Untuk menguji secara opsional apakah Anda sudah memiliki sesi aktif, jalankan AWS CLI perintah berikut.

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

Jika sesi Anda aktif, respons terhadap perintah ini melaporkan akun Pusat Identitas IAM dan set izin yang dikonfigurasi dalam `config` file bersama.

> **Note**
>
> Jika Anda sudah memiliki sesi portal AWS akses aktif dan menjalankan`aws sso login`, Anda tidak akan diminta untuk memberikan kredensi.
> Proses masuk mungkin meminta Anda untuk mengizinkan AWS CLI akses ke data Anda. Karena AWS CLI dibangun di atas SDK untuk Python, pesan izin mungkin berisi variasi nama. `botocore`

Informasi otentikasi lebih lanjut

Pengguna manusia, juga dikenal sebagai identitas manusia, adalah orang, administrator, pengembang, operator, dan konsumen aplikasi Anda. Mereka harus memiliki identitas untuk mengakses AWS lingkungan dan aplikasi Anda. Pengguna manusia yang merupakan anggota organisasi Anda - itu berarti Anda, pengembang - dikenal sebagai identitas tenaga kerja.

Gunakan kredensi sementara saat mengakses. AWS Anda dapat menggunakan penyedia identitas bagi pengguna manusia Anda untuk menyediakan akses gabungan ke AWS akun dengan mengambil peran, yang menyediakan kredensi sementara. Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center (IAM Identity Center) untuk mengelola akses ke akun Anda dan izin dalam akun tersebut. Untuk alternatif lainnya, lihat yang berikut ini:

- Untuk mempelajari lebih lanjut tentang praktik terbaik, lihat Praktik terbaik keamanan di IAM di Panduan Pengguna IAM.
- Untuk membuat AWS kredensi jangka pendek, lihat Kredenal Keamanan Sementara di Panduan Pengguna IAM.
- Untuk mempelajari SDK lain untuk penyedia kredensi, lihat Penyedia JavaScript kredensi standar di Panduan Referensi Alat dan SDK lainnya.AWS SDKs

Mengatur Kredensial

AWS menggunakan kredensi untuk mengidentifikasi siapa yang memanggil layanan dan apakah akses ke sumber daya yang diminta diizinkan.

Baik berjalan di browser web atau di server Node.js, JavaScript kode Anda harus mendapatkan kredensi yang valid sebelum dapat mengakses layanan melalui API. Kredensial dapat diatur secara global pada objek konfigurasi, menggunakan`AWS.Config`, atau per layanan, dengan meneruskan kredensi langsung ke objek layanan.

Ada beberapa cara untuk mengatur kredensil yang berbeda antara Node.js dan JavaScript di browser web. Topik di bagian ini menjelaskan cara mengatur kredensional di Node.js atau browser web. Dalam setiap kasus, opsi disajikan dalam urutan yang disarankan.

Praktik Terbaik untuk Kredensial

Menetapkan kredensil dengan benar memastikan bahwa aplikasi atau skrip browser Anda dapat mengakses layanan dan sumber daya yang diperlukan sambil meminimalkan paparan terhadap masalah keamanan yang dapat memengaruhi aplikasi penting misi atau membahayakan data sensitif.

Prinsip penting untuk diterapkan saat menetapkan kredensional adalah selalu memberikan hak istimewa paling sedikit yang diperlukan untuk tugas Anda. Lebih aman untuk memberikan izin minimal pada sumber daya Anda dan menambahkan izin lebih lanjut sesuai kebutuhan, daripada memberikan izin yang melebihi hak istimewa paling sedikit dan, sebagai hasilnya, diminta untuk memperbaiki masalah keamanan yang mungkin Anda temukan nanti. Misalnya, kecuali Anda perlu membaca dan menulis sumber daya individual, seperti objek di bucket Amazon S3 atau tabel DynamoDB, setel izin tersebut hanya untuk dibaca.

Untuk informasi selengkapnya tentang pemberian hak istimewa paling sedikit, lihat bagian Hibah Hak Istimewa Paling Sedikit dari topik Praktik Terbaik di Panduan Pengguna IAM.

> **Warning**
>
> Meskipun dimungkinkan untuk melakukannya, kami menyarankan Anda untuk tidak menggunakan kredensi kode keras di dalam aplikasi atau skrip browser. Kredensi hard coding menimbulkan risiko mengekspos informasi sensitif.

Untuk informasi selengkapnya tentang cara mengelola kunci akses, lihat Praktik Terbaik untuk Mengelola Kunci AWS Akses di Referensi Umum AWS.

Topik

- Mengatur Kredensial di Node.js

- Menyetel Kredensial di Browser Web

Mengatur Kredensial di Node.js

Ada beberapa cara di Node.js untuk memasok kredensialmu ke SDK. Beberapa di antaranya lebih aman dan yang lain memberikan kenyamanan yang lebih besar saat mengembangkan aplikasi. Saat mendapatkan kredensi di Node.js, berhati-hatilah untuk mengandalkan lebih dari satu sumber seperti variabel lingkungan dan file JSON yang Anda muat. Anda dapat mengubah izin di mana kode Anda berjalan tanpa menyadari perubahan telah terjadi.

Berikut adalah cara-cara Anda dapat memberikan kredensi Anda dalam urutan rekomendasi:

1. Dimuat dari peran AWS Identity and Access Management (IAM) untuk Amazon EC2
2. Dimuat dari file kredensial bersama () `~/.aws/credentials`
3. Dimuat dari variabel lingkungan
4. Dimuat dari file JSON pada disk
5. Kelas penyedia kredensial-lainnya yang disediakan oleh SDK JavaScript

Jika lebih dari satu sumber kredensi tersedia untuk SDK, prioritas default pemilihan adalah sebagai berikut:

1. Kredensional yang ditetapkan secara eksplisit melalui konstruktor layanan-klien
2. Variabel-variabel lingkungan
3. Berkas kredensial bersama
4. Kredensi dimuat dari penyedia kredensi ECS (jika ada)
5. Kredensial yang diperoleh dengan menggunakan proses kredensi yang ditentukan dalam file AWS konfigurasi bersama atau file kredensial bersama. Untuk informasi selengkapnya, lihat the section called “Kredensial menggunakan Proses Kredenal yang Dikonfigurasi”.
6. Kredenal dimuat dari AWS IAM menggunakan penyedia kredensional EC2 instans Amazon (jika dikonfigurasi dalam metadata instans)

Untuk informasi selengkapnya, lihat `Class: AWS.Credentials`dan `Class: AWS.CredentialProviderChain`di referensi API.

> **Warning**
>
> Meskipun dimungkinkan untuk melakukannya, kami tidak menyarankan hard-coding AWS kredensional Anda dalam aplikasi Anda. Kredensi hard-coding menimbulkan risiko mengekspos ID kunci akses dan kunci akses rahasia Anda.

Topik di bagian ini menjelaskan cara memuat kredensi ke dalam Node.js.

Topik

- Memuat Kredensi di Node.js dari peran IAM untuk Amazon EC2
- Memuat Kredensi untuk Fungsi Lambda Node.js
- Memuat Kredensial di Node.js dari File Kredensial Bersama
- Memuat Kredensi di Node.js dari Variabel Lingkungan
- Memuat Kredensial di Node.js dari File JSON
- Memuat Kredensial di Node.js menggunakan Proses Kredenal yang Dikonfigurasi

Memuat Kredensi di Node.js dari peran IAM untuk Amazon EC2

Jika Anda menjalankan aplikasi Node.js di EC2 instans Amazon, Anda dapat memanfaatkan peran IAM untuk Amazon EC2 agar secara otomatis memberikan kredensi ke instans. Jika Anda mengonfigurasi instans Anda untuk menggunakan peran IAM, SDK secara otomatis memilih kredenal IAM untuk aplikasi Anda, sehingga tidak perlu menyediakan kredenal secara manual.

Untuk informasi selengkapnya tentang menambahkan peran IAM ke EC2 instans Amazon, lihat Menggunakan peran IAM untuk EC2 instans Amazon di Panduan Referensi Alat AWS SDKs dan Alat.

Memuat Kredensi untuk Fungsi Lambda Node.js

Saat Anda membuat AWS Lambda fungsi, Anda harus membuat peran IAM khusus yang memiliki izin untuk menjalankan fungsi tersebut. Peran ini disebut peran eksekusi. Saat menyiapkan fungsi Lambda, Anda harus menentukan peran IAM yang Anda buat sebagai peran eksekusi yang sesuai.

Peran eksekusi menyediakan fungsi Lambda dengan kredenal yang dibutuhkan untuk menjalankan dan memanggil layanan web lainnya. Akibatnya, Anda tidak perlu memberikan kredensi ke kode Node.js yang Anda tulis dalam fungsi Lambda.

Untuk informasi selengkapnya tentang membuat peran eksekusi Lambda, lihat Mengelola Izin: Menggunakan Peran IAM (Peran Eksekusi) di Panduan Pengembang.AWS Lambda

Memuat Kredensial di Node.js dari File Kredensial Bersama

Anda dapat menyimpan data AWS kredensional Anda dalam file bersama yang digunakan oleh SDKs dan antarmuka baris perintah. Saat SDK JavaScript dimuat, SDK secara otomatis mencari file kredensial bersama, yang diberi nama "kredensial". Tempat Anda menyimpan file kredensi bersama tergantung pada sistem operasi Anda:

- File kredensial bersama di Linux, Unix, dan macOS: `~/.aws/credentials`
- File kredensial bersama di Windows: `C:\Users\USER_NAME\.aws\credentials`

Jika Anda belum memiliki file kredensial bersama, lihat. Otentikasi SDK dengan AWS Setelah Anda mengikuti instruksi tersebut, Anda akan melihat teks yang mirip dengan yang berikut di file kredensial, di mana *<YOUR_ACCESS_KEY_ID>* ID kunci akses Anda dan *<YOUR_SECRET_ACCESS_KEY>* merupakan kunci akses rahasia Anda:

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

Untuk contoh yang menunjukkan file ini sedang digunakan, lihatMemulai di Node.js.

Judul `[default]` bagian menentukan profil default dan nilai-nilai terkait untuk kredensional. Anda dapat membuat profil tambahan dalam file konfigurasi bersama yang sama, masing-masing dengan informasi kredensialnya sendiri. Contoh berikut menunjukkan file konfigurasi dengan profil default dan dua profil tambahan:

```
[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>
```

Secara default, SDK memeriksa variabel `AWS_PROFILE` lingkungan untuk menentukan profil mana yang akan digunakan. Jika `AWS_PROFILE` variabel tidak disetel di lingkungan Anda, SDK akan menggunakan kredensional untuk profil tersebut. `[default]` Untuk menggunakan salah satu profil alternatif, atur atau ubah nilai variabel `AWS_PROFILE` lingkungan. Misalnya, mengingat file konfigurasi yang ditunjukkan di atas, untuk menggunakan kredensi dari akun kerja, atur variabel `AWS_PROFILE` lingkungan ke `work-account` (yang sesuai untuk sistem operasi Anda).

> **Note**
>
> Saat mengatur variabel lingkungan, pastikan untuk mengambil tindakan yang tepat setelahnya (sesuai dengan kebutuhan sistem operasi Anda) untuk membuat variabel tersedia di shell atau lingkungan perintah.

Setelah menyetel variabel lingkungan (jika diperlukan), Anda dapat menjalankan JavaScript file yang menggunakan SDK, seperti misalnya, file bernama`script.js`.

```
$ node script.js
```

Anda juga dapat secara eksplisit memilih profil yang digunakan oleh SDK, baik dengan menyetel `process.env.AWS_PROFILE` sebelum memuat SDK, atau dengan memilih penyedia kredensi seperti yang ditunjukkan pada contoh berikut:

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

Memuat Kredensi di Node.js dari Variabel Lingkungan

SDK secara otomatis mendeteksi AWS kredensial yang ditetapkan sebagai variabel di lingkungan Anda dan menggunakannya untuk permintaan SDK, sehingga menghilangkan kebutuhan untuk mengelola kredensi dalam aplikasi Anda. Variabel lingkungan yang Anda tetapkan untuk memberikan kredensialnya adalah:

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

Untuk detail selengkapnya tentang pengaturan variabel lingkungan, lihat Dukungan variabel lingkungan di Panduan Referensi Alat AWS SDKs dan Alat.

Memuat Kredensial di Node.js dari File JSON

Anda dapat memuat konfigurasi dan kredensional dari dokumen JSON pada disk menggunakan. `AWS.config.loadFromPath` Jalur yang ditentukan relatif terhadap direktori kerja saat ini dari proses Anda. Misalnya, untuk memuat kredensi dari `'config.json'` file dengan konten berikut:

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

Kemudian gunakan kode berikut:

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

> **Note**
>
> Memuat data konfigurasi dari dokumen JSON akan me-reset semua data konfigurasi yang ada. Tambahkan data konfigurasi tambahan setelah menggunakan teknik ini. Memuat kredensi dari dokumen JSON tidak didukung dalam skrip browser.

Memuat Kredensial di Node.js menggunakan Proses Kredenal yang Dikonfigurasi

Anda dapat mencari kredensi dengan menggunakan metode yang tidak dibangun ke dalam SDK. Untuk melakukannya, tentukan proses kredensi dalam file AWS config bersama atau file kredensial bersama. Jika variabel `AWS_SDK_LOAD_CONFIG` lingkungan disetel ke nilai apa pun, SDK akan lebih memilih proses yang ditentukan dalam file konfigurasi daripada proses yang ditentukan dalam file kredensial (jika ada).

Untuk detail tentang menentukan proses kredensi dalam file AWS konfigurasi bersama atau file kredensial bersama, lihat Referensi AWS CLI Perintah, khususnya informasi tentang Sumber Kredensial Dari Proses Eksternal.

Untuk informasi tentang penggunaan variabel `AWS_SDK_LOAD_CONFIG` lingkungan, lihat the section called “Menggunakan File Config Bersama” di dokumen ini.

Menyetel Kredensial di Browser Web

Ada beberapa cara untuk memasok kredensional Anda ke SDK dari skrip browser. Beberapa di antaranya lebih aman dan yang lain memberikan kenyamanan yang lebih besar saat mengembangkan skrip. Berikut adalah cara-cara Anda dapat memberikan kredensi Anda dalam urutan rekomendasi:

1. Menggunakan Identitas Amazon Cognito untuk mengautentikasi pengguna dan menyediakan kredensional
2. Menggunakan identitas federasi web
3. Hard code dalam script

> **Warning**
>
> Kami tidak menyarankan hard coding AWS kredensional Anda dalam skrip Anda. Kredensi pengkodean keras menimbulkan risiko mengekspos ID kunci akses dan kunci akses rahasia Anda.

Topik

- Menggunakan Identitas Amazon Cognito untuk Mengautentikasi Pengguna
- Menggunakan Identitas Federasi Web untuk Mengautentikasi Pengguna
- Contoh Identitas Federasi Web

Menggunakan Identitas Amazon Cognito untuk Mengautentikasi Pengguna

Cara yang disarankan untuk mendapatkan AWS kredensi untuk skrip browser Anda adalah dengan menggunakan objek kredensi Identitas Amazon Cognito,. `AWS.CognitoIdentityCredentials` Amazon Cognito memungkinkan otentikasi pengguna melalui penyedia identitas pihak ketiga.

Untuk menggunakan Identitas Amazon Cognito, Anda harus terlebih dahulu membuat kumpulan identitas di konsol Amazon Cognito. Kumpulan identitas mewakili grup identitas yang disediakan aplikasi Anda kepada pengguna Anda. Identitas yang diberikan kepada pengguna secara unik mengidentifikasi setiap akun pengguna. Identitas Amazon Cognito identitas bukan kredensial. Mereka ditukar dengan kredensi menggunakan dukungan federasi identitas web di AWS Security Token Service ().AWS STS

Amazon Cognito membantu Anda mengelola abstraksi identitas di beberapa penyedia identitas dengan objek. `AWS.CognitoIdentityCredentials` Identitas yang dimuat kemudian ditukar dengan kredensi di. AWS STS

Mengonfigurasi Objek Kredensial Identitas Amazon Cognito

Jika Anda belum membuatnya, buat kumpulan identitas untuk digunakan dengan skrip browser Anda di konsol Amazon Cognito sebelum Anda mengonfigurasi. `AWS.CognitoIdentityCredentials` Buat dan kaitkan peran IAM yang diautentikasi dan tidak diautentikasi untuk kumpulan identitas Anda.

Pengguna yang tidak diautentikasi tidak meminta agar identitas mereka diverifikasi, sehingga peran ini sesuai untuk pengguna tamu aplikasi Anda atau jika pengguna meminta agar identitas mereka diverifikasi dan itu tidak dipermasalahkan. Pengguna yang diautentikasi masuk ke aplikasi Anda melalui penyedia identitas pihak ketiga yang memverifikasi identitas mereka. Pastikan Anda menjangkau izin sumber daya dengan tepat sehingga Anda tidak memberikan akses kepada mereka dari pengguna yang tidak terautentikasi.

Setelah mengonfigurasi kumpulan identitas dengan penyedia identitas yang dilampirkan, Anda dapat menggunakannya `AWS.CognitoIdentityCredentials` untuk mengautentikasi pengguna. Untuk mengkonfigurasi kredensial aplikasi Anda agar dapat menggunakan `AWS.CognitoIdentityCredentials`, atur properti `credentials` baik `AWS.Config` atau konfigurasi per layanan. Contoh berikut menggunakan `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'
  }
});
```

Properti opsional `Logins` adalah peta nama penyedia identitas untuk token identitas bagi penyedia tersebut. Bagaimana Anda bisa mendapatkan token dari penyedia identitas Anda tergantung pada penyedia yang Anda gunakan. Misalnya, jika Facebook adalah salah satu penyedia identitas Anda, Anda dapat menggunakan fungsi `FB.login` dari SDK Facebook untuk mendapatkan token penyedia identitas:

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

Mengalihkan Pengguna yang Tidak Diautentikasi ke Pengguna yang Diautentikasi

Amazon Cognito mendukung pengguna yang diautentikasi dan tidak diautentikasi. Pengguna yang tidak diautentikasi menerima akses ke sumber daya Anda meskipun mereka tidak masuk dengan penyedia identitas Anda. Tingkat akses ini berguna untuk menampilkan konten kepada pengguna sebelum masuk. Setiap pengguna yang tidak diautentikasi memiliki identitas unik di Amazon Cognito meskipun mereka belum masuk dan diautentikasi secara individual.

Pengguna Awalnya Tidak Diautentikasi

Pengguna biasanya memulai dengan peran yang tidak diautentikasi, di mana Anda menetapkan properti kredensial objek konfigurasi Anda tanpa properti. `Logins` Dalam kasus ini, konfigurasi default Anda mungkin terlihat seperti berikut ini:

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

Beralih ke Pengguna Terautentikasi

Ketika pengguna yang tidak diautentikasi masuk ke penyedia identitas dan Anda memiliki token, Anda dapat mengalihkan pengguna dari yang tidak diautentikasi ke otentikasi dengan memanggil fungsi kustom yang memperbarui objek kredensi dan menambahkan 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;
}
```

Anda juga dapat membuat `CognitoIdentityCredentials` objek. Jika ya, Anda harus mengatur ulang properti kredensial dari objek layanan yang ada yang Anda buat. Objek layanan dibaca dari konfigurasi global hanya pada inisialisasi objek.

Untuk informasi selengkapnya tentang `CognitoIdentityCredentials` objek, lihat `AWS.CognitoIdentityCredentials`di Referensi AWS SDK untuk JavaScript API.

Menggunakan Identitas Federasi Web untuk Mengautentikasi Pengguna

Anda dapat langsung mengkonfigurasi penyedia identitas individu untuk mengakses AWS sumber daya menggunakan federasi identitas web. AWS saat ini mendukung otentikasi pengguna menggunakan federasi identitas web melalui beberapa penyedia identitas:

- Login with Amazon
- Login Facebook
- Masuk Google

Anda harus terlebih dahulu mendaftarkan aplikasi Anda ke penyedia yang didukung aplikasi Anda. Selanjutnya, buat peran IAM dan atur izin untuk itu. Peran IAM yang Anda buat kemudian digunakan untuk memberikan izin yang Anda konfigurasikan untuk itu melalui penyedia identitas masing-masing. Misalnya, Anda dapat mengatur peran yang memungkinkan pengguna yang masuk melalui Facebook memiliki akses baca ke bucket Amazon S3 tertentu yang Anda kontrol.

Setelah Anda memiliki peran IAM dengan hak istimewa yang dikonfigurasi dan aplikasi yang terdaftar dengan penyedia identitas pilihan Anda, Anda dapat mengatur SDK untuk mendapatkan kredensi untuk peran IAM menggunakan kode pembantu, sebagai berikut:

```
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
});
```

Nilai dalam `ProviderId` parameter tergantung pada penyedia identitas yang ditentukan. Nilai dalam `WebIdentityToken` parameter adalah token akses yang diambil dari login yang berhasil dengan penyedia identitas. Untuk informasi selengkapnya tentang cara mengonfigurasi dan mengambil token akses untuk setiap penyedia identitas, lihat dokumentasi untuk penyedia identitas.

Langkah 1: Mendaftar dengan Penyedia Identitas

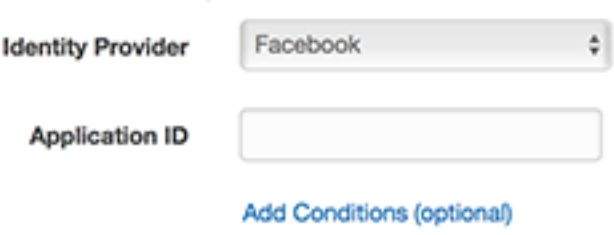
Untuk memulai, daftarkan aplikasi dengan penyedia identitas yang Anda pilih untuk didukung. Anda akan diminta untuk memberikan informasi yang mengidentifikasi aplikasi Anda dan mungkin penulisnya. Ini memastikan bahwa penyedia identitas tahu siapa yang menerima informasi pengguna mereka. Dalam setiap kasus, penyedia identitas akan mengeluarkan ID aplikasi yang Anda gunakan untuk mengonfigurasi peran pengguna.

Langkah 2: Membuat Peran IAM untuk Penyedia Identitas

Setelah Anda mendapatkan ID aplikasi dari penyedia identitas, buka konsol IAM di https://console.aws.amazon.com/iam/untuk membuat peran IAM baru.

Untuk membuat peran IAM untuk penyedia identitas

1. Buka bagian Peran konsol dan kemudian pilih Buat Peran Baru.
2. Ketik nama untuk peran baru yang membantu Anda melacak penggunaannya, seperti**`facebookIdentity`**, lalu pilih Langkah Berikutnya.
3. Di Pilih Jenis Peran, pilih Peran untuk Akses Penyedia Identitas.
4. Untuk memberikan akses ke penyedia identitas web, pilih Pilih.
5. Dari daftar Penyedia Identitas, pilih penyedia identitas yang ingin Anda gunakan untuk peran IAM ini.



6. Ketik ID aplikasi yang disediakan oleh penyedia identitas di ID Aplikasi dan kemudian pilih Langkah Berikutnya.

7. Konfigurasikan izin untuk sumber daya yang ingin Anda paparkan, yang memungkinkan akses ke operasi tertentu pada sumber daya tertentu. Untuk informasi selengkapnya tentang izin IAM, lihat Ringkasan Izin AWS IAM di Panduan Pengguna IAM. Tinjau dan, jika perlu, sesuaikan hubungan kepercayaan peran, lalu pilih Langkah Berikutnya.
8. Lampirkan kebijakan tambahan yang Anda butuhkan dan kemudian pilih Langkah Berikutnya. Untuk informasi lebih lanjut tentang kebijakan IAM, lihat Gambaran umum Kebijakan IAM dalam Panduan Pengguna IAM.
9. Tinjau peran baru dan kemudian pilih Buat Peran.

Anda dapat memberikan batasan lain untuk peran tersebut, seperti melingkupinya ke pengguna tertentu. IDs Jika peran memberikan izin menulis ke sumber daya Anda, pastikan Anda mencakupkan peran dengan benar kepada pengguna dengan hak istimewa yang benar, jika tidak, setiap pengguna dengan identitas Amazon, Facebook, atau Google akan dapat memodifikasi sumber daya dalam aplikasi Anda.

Untuk informasi selengkapnya tentang penggunaan federasi identitas web di IAM, lihat Tentang Federasi Identitas Web di Panduan Pengguna IAM.

Langkah 3: Mendapatkan Token Akses Penyedia Setelah Login

Siapkan tindakan login untuk aplikasi Anda dengan menggunakan SDK penyedia identitas. Anda dapat mengunduh dan menginstal JavaScript SDK dari penyedia identitas yang memungkinkan login pengguna, menggunakan salah satu OAuth atau OpenID. Untuk informasi tentang cara mengunduh dan menyiapkan kode SDK di aplikasi Anda, lihat dokumentasi SDK untuk penyedia identitas Anda:

- Login with Amazon
- Login Facebook
- Masuk Google

Langkah 4: Memperoleh Kredensi Sementara

Setelah aplikasi, peran, dan izin sumber daya dikonfigurasi, tambahkan kode ke aplikasi Anda untuk mendapatkan kredensi sementara. Kredensi ini disediakan melalui federasi identitas web yang AWS Security Token Service menggunakan. Pengguna masuk ke penyedia identitas, yang mengembalikan token akses. Siapkan `AWS.WebIdentityCredentials` objek menggunakan ARN untuk peran IAM yang Anda buat untuk penyedia identitas ini:

```
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
});
```

Objek layanan yang dibuat selanjutnya akan memiliki kredensional yang tepat. Objek yang dibuat sebelum menyetel `AWS.config.credentials` properti tidak akan memiliki kredensi saat ini.

Anda juga dapat membuat `AWS.WebIdentityCredentials` sebelum mengambil token akses. Ini memungkinkan Anda untuk membuat objek layanan yang bergantung pada kredensi sebelum memuat token akses. Untuk melakukan ini, buat objek kredensial tanpa parameter: `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;
```

Kemudian atur `WebIdentityToken` callback dari SDK penyedia identitas yang berisi token akses:

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

Contoh Identitas Federasi Web

Berikut adalah beberapa contoh menggunakan identitas federasi web untuk mendapatkan kredensi di browser. JavaScript Contoh-contoh ini harus dijalankan dari skema host http://atau https://untuk memastikan penyedia identitas dapat mengarahkan ulang ke aplikasi Anda.

Login with Amazon Contoh

Kode berikut menunjukkan cara menggunakan Login with Amazon sebagai penyedia identitas.

```
<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>
```

Contoh Login Facebook

Kode berikut menunjukkan cara menggunakan Facebook Login sebagai penyedia identitas:

```
<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>
```

Contoh Masuk Google+

Kode berikut menunjukkan cara menggunakan Google+ Sign-in sebagai penyedia identitas. Token akses yang digunakan untuk federasi identitas web dari Google disimpan `response.id_token` bukan `access_token` seperti penyedia identitas lainnya.

```
<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>
```

Mengunci Versi API

AWS layanan memiliki nomor versi API untuk melacak kompatibilitas API. Versi API dalam AWS layanan diidentifikasi oleh string tanggal yang `YYYY-mm-dd` diformat. Misalnya, versi API saat ini untuk Amazon S3 adalah. `2006-03-01`

Kami merekomendasikan untuk mengunci versi API untuk layanan jika Anda mengandalkannya dalam kode produksi. Ini dapat mengisolasi aplikasi Anda dari perubahan layanan yang dihasilkan dari pembaruan ke SDK. Jika Anda tidak menentukan versi API saat membuat objek layanan, SDK menggunakan versi API terbaru secara default. Hal ini dapat menyebabkan aplikasi Anda mereferensikan API yang diperbarui dengan perubahan yang berdampak negatif pada aplikasi Anda.

Untuk mengunci versi API yang Anda gunakan untuk layanan, teruskan `apiVersion` parameter saat membuat objek layanan. Dalam contoh berikut, objek `AWS.DynamoDB` layanan yang baru dibuat dikunci ke versi `2011-12-05` API:

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

Anda dapat mengonfigurasi serangkaian versi API layanan secara global dengan menentukan `apiVersions` parameter di`AWS.Config`. Misalnya, untuk menyetel versi tertentu dari DynamoDB dan EC2 APIs Amazon bersama dengan Amazon Redshift API saat ini, tetapkan sebagai berikut: `apiVersions`

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

Mendapatkan Versi API

Untuk mendapatkan versi API untuk layanan, lihat bagian Mengunci Versi API di halaman referensi layanan, seperti https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html untuk Amazon S3.

Pertimbangan Node.js

Meskipun kode Node.js adalah JavaScript, menggunakan AWS SDK untuk JavaScript dalam Node.js dapat berbeda dari menggunakan SDK dalam skrip browser. Beberapa metode API bekerja di

Node.js tetapi tidak dalam skrip browser, serta sebaliknya. Dan berhasil menggunakan beberapa APIs tergantung pada keakraban Anda dengan pola pengkodean Node.js umum, seperti mengimpor dan menggunakan modul Node.js lainnya seperti modul. `File System (fs)`

Menggunakan Modul Node.js Built-In

Node.js menyediakan koleksi modul bawaan yang dapat Anda gunakan tanpa menginstalnya. Untuk menggunakan modul ini, buat objek dengan `require` metode untuk menentukan nama modul. Misalnya, untuk menyertakan modul HTTP bawaan, gunakan yang berikut ini.

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

Memanggil metode modul seolah-olah mereka adalah metode dari objek itu. Misalnya, berikut adalah kode yang membaca 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
  }
});
```

Untuk daftar lengkap semua modul bawaan yang disediakan Node.js, lihat Dokumentasi Node.js v6.11.1 di situs web Node.js.

Menggunakan Paket NPM

Selain modul bawaan, Anda juga dapat menyertakan dan menggabungkan kode pihak ketiga dari npm, manajer paket Node.js. Ini adalah repositori paket Node.js open source dan antarmuka baris perintah untuk menginstal paket-paket tersebut. Untuk informasi selengkapnya tentang npm dan daftar paket yang tersedia saat ini, lihat https://www.npmjs.com. Anda juga dapat mempelajari tentang paket Node.js tambahan yang dapat Anda gunakan di sini GitHub.

Salah satu contoh paket npm yang dapat Anda gunakan AWS SDK untuk JavaScript adalah`browserify`. Untuk detailnya, lihat Membangun SDK sebagai Dependensi dengan Browserify. Contoh lain adalah`webpack`. Untuk detailnya, lihat Bundling Aplikasi dengan Webpack.

Topik

- Mengkonfigurasi MaxSockets di Node.js
- Menggunakan Kembali Koneksi dengan Keep-Alive di Node.js
- Mengkonfigurasi Proxy untuk Node.js
- Mendaftarkan Bundel Sertifikat di Node.js

Mengkonfigurasi MaxSockets di Node.js

Di Node.js, Anda dapat mengatur jumlah maksimum koneksi per asal. Jika `maxSockets` diatur, klien HTTP tingkat rendah mengantri permintaan dan menetapkannya ke soket saat tersedia.

Ini memungkinkan Anda menetapkan batas atas pada jumlah permintaan bersamaan ke asal tertentu pada suatu waktu. Menurunkan nilai ini dapat mengurangi jumlah kesalahan pelambatan atau batas waktu yang diterima. Namun, itu juga dapat meningkatkan penggunaan memori karena permintaan antri sampai soket tersedia.

Contoh berikut menunjukkan cara mengatur `maxSockets` untuk semua objek layanan yang Anda buat. Contoh ini memungkinkan hingga 25 koneksi bersamaan ke setiap titik akhir layanan.

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

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

Hal yang sama dapat dilakukan per layanan.

```
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
   }
});
```

Saat menggunakan default`https`, SDK mengambil `maxSockets` nilai dari. `globalAgent` Jika `maxSockets` nilai tidak didefinisikan atau tidak`Infinity`, SDK mengasumsikan `maxSockets` nilai 50.

Untuk informasi selengkapnya tentang pengaturan `maxSockets` di Node.js, lihat dokumentasi online Node.js.

Menggunakan Kembali Koneksi dengan Keep-Alive di Node.js

Secara default, agen HTTP/HTTPS Node.js default membuat koneksi TCP baru untuk setiap permintaan baru. Untuk menghindari biaya membuat koneksi baru, Anda dapat menggunakan kembali koneksi yang ada.

Untuk operasi jangka pendek, seperti query DynamoDB, overhead latensi pengaturan koneksi TCP mungkin lebih besar daripada operasi itu sendiri. Selain itu, karena enkripsi DynamoDB saat istirahat terintegrasi AWS dengan KMS, Anda mungkin mengalami latensi dari database yang harus membuat kembali AWS KMS entri cache baru untuk setiap operasi.

Cara termudah untuk mengonfigurasi SDK JavaScript agar dapat menggunakan kembali koneksi TCP adalah dengan mengatur variabel `AWS_NODEJS_CONNECTION_REUSE_ENABLED` lingkungan ke. 1 Fitur ini ditambahkan dalam rilis 2.463.0.

Atau, Anda dapat mengatur `keepAlive` properti agen HTTP atau HTTPS yang disetel ke`true`, seperti yang ditunjukkan pada contoh berikut.

```
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
  }
});
```

Contoh berikut menunjukkan bagaimana mengatur hanya `keepAlive` untuk klien 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
  }
});
```

Jika `keepAlive` diaktifkan, Anda juga dapat mengatur penundaan awal untuk paket TCP Keep-Alive`keepAliveMsecs`, yang secara default adalah 1000ms. Lihat dokumentasi Node.js untuk detailnya.

Mengkonfigurasi Proxy untuk Node.js

Jika Anda tidak dapat terhubung langsung ke internet, SDK untuk JavaScript mendukung penggunaan proxy HTTP atau HTTPS melalui agen HTTP pihak ketiga, seperti agen proxy. Untuk menginstal proxy-agent, ketik berikut ini di baris perintah.

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

Jika Anda memutuskan untuk menggunakan proxy yang berbeda, pertama-tama ikuti instruksi instalasi dan konfigurasi untuk proxy tersebut. Untuk menggunakan ini atau proxy pihak ketiga lainnya dalam aplikasi Anda, Anda harus mengatur `httpOptions` properti `AWS.Config` untuk menentukan proxy yang Anda pilih. Contoh ini menunjukkan `proxy-agent`.

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

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

Untuk informasi selengkapnya tentang pustaka proxy lainnya, lihat npm, pengelola paket Node.js.

Mendaftarkan Bundel Sertifikat di Node.js

Penyimpanan kepercayaan default untuk Node.js menyertakan sertifikat yang diperlukan untuk mengakses AWS layanan. Dalam beberapa kasus, mungkin lebih baik untuk menyertakan hanya satu set sertifikat tertentu.

Dalam contoh ini, sertifikat khusus pada disk digunakan untuk membuat `https.Agent` yang menolak koneksi kecuali sertifikat yang ditunjuk disediakan. Yang baru `https.Agent` dibuat kemudian digunakan untuk memperbarui konfigurasi 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
    })
  }
});
```

Pertimbangan Skrip Browser

Topik berikut menjelaskan pertimbangan khusus untuk menggunakan skrip AWS SDK untuk JavaScript di browser.

Topik

- Membangun SDK untuk Browser

- Berbagi Sumber Daya Lintas Orisinil (CORS)

Membangun SDK untuk Browser

SDK untuk JavaScript disediakan sebagai JavaScript file dengan dukungan yang disertakan untuk serangkaian layanan default. File ini biasanya dimuat ke skrip browser menggunakan `<script>` tag yang mereferensikan paket SDK yang dihosting. Namun, Anda mungkin memerlukan dukungan untuk layanan selain set default atau perlu menyesuaikan SDK.

Jika Anda bekerja dengan SDK di luar lingkungan yang memberlakukan CORS di browser Anda dan jika Anda ingin akses ke semua layanan yang disediakan oleh SDK for JavaScript, Anda dapat membuat salinan khusus SDK secara lokal dengan mengkloning repositori dan menjalankan alat build yang sama yang membangun versi default SDK yang dihosting. Bagian berikut menjelaskan langkah-langkah untuk membangun SDK dengan layanan tambahan dan versi API.

Topik

- Menggunakan SDK Builder untuk Membangun SDK JavaScript
- Menggunakan CLI untuk Membangun SDK untuk JavaScript
- Membangun Layanan Tertentu dan Versi API
- Membangun SDK sebagai Dependensi dengan Browserify

Menggunakan SDK Builder untuk Membangun SDK JavaScript

Cara termudah untuk membuat build Anda sendiri AWS SDK untuk JavaScript adalah dengan menggunakan aplikasi web pembuat SDK di https://sdk.amazonaws.com/builder/js. Gunakan pembuat SDK untuk menentukan layanan, dan versi API-nya, untuk disertakan dalam build Anda.

Pilih Pilih semua layanan atau pilih Pilih layanan default sebagai titik awal dari mana Anda dapat menambah atau menghapus layanan. Pilih Pengembangan untuk kode yang lebih mudah dibaca atau pilih Diperkecil untuk membuat build yang diperkecil untuk diterapkan. Setelah memilih layanan dan versi yang akan disertakan, pilih Build to build dan download SDK kustom Anda.

Menggunakan CLI untuk Membangun SDK untuk JavaScript

Untuk membangun SDK untuk JavaScript menggunakan AWS CLI, Anda harus terlebih dahulu mengkloning repositori Git yang berisi sumber SDK. Anda harus menginstal Git dan Node.js di komputer Anda.

Pertama, kloning repositori dari GitHub dan ubah direktori ke direktori:

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

Setelah Anda mengkloning repositori, unduh modul dependensi untuk SDK dan alat build:

```
npm install
```

Anda sekarang dapat membuat versi paket SDK.

Membangun dari Command Line

Alat pembangun ada di`dist-tools/browser-builder.js`. Jalankan skrip ini dengan mengetik:

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

Perintah ini membangun file aws-sdk.js. File ini tidak terkompresi. Secara default paket ini hanya mencakup serangkaian layanan standar.

Meminimalkan Output Build

Untuk mengurangi jumlah data pada jaringan, JavaScript file dapat dikompresi melalui proses yang disebut minifikasi. Minifikasi menghapus komentar, spasi yang tidak perlu, dan karakter lain yang membantu keterbacaan manusia tetapi itu tidak memengaruhi eksekusi kode. Alat pembangun dapat menghasilkan output yang tidak terkompresi atau diperkecil. Untuk memperkecil keluaran build Anda, setel variabel `MINIFY` lingkungan:

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

Membangun Layanan Tertentu dan Versi API

Anda dapat memilih layanan mana yang akan dibangun ke dalam SDK. Untuk memilih layanan, tentukan nama layanan, dibatasi oleh koma, sebagai parameter. Misalnya, untuk hanya membangun Amazon S3 dan Amazon EC2, gunakan perintah berikut:

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

Anda juga dapat memilih versi API tertentu dari build layanan dengan menambahkan nama versi setelah nama layanan. Misalnya, untuk membangun kedua versi API Amazon DynamoDB, gunakan perintah berikut:

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

Pengidentifikasi layanan dan versi API tersedia dalam file konfigurasi khusus layanan di/. https://github.com/aws/ aws-sdk-js tree/master/apis

Membangun Semua Layanan

Anda dapat membangun semua layanan dan versi API dengan menyertakan `all` parameter:

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

Membangun Layanan Khusus

Untuk menyesuaikan kumpulan layanan yang dipilih yang disertakan dalam build, teruskan variabel `AWS_SERVICES` lingkungan ke perintah Browserify yang berisi daftar layanan yang Anda inginkan. Contoh berikut membangun layanan Amazon EC2, Amazon S3, dan DynamoDB.

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

Membangun SDK sebagai Dependensi dengan Browserify

Node.js memiliki mekanisme berbasis modul untuk memasukkan kode dan fungsionalitas dari pengembang pihak ketiga. Pendekatan modular ini tidak didukung secara native dengan JavaScript berjalan di browser web. Namun, dengan alat yang disebut Browserify, Anda dapat menggunakan pendekatan modul Node.js dan menggunakan modul yang ditulis untuk Node.js di browser. Browserify membangun dependensi modul untuk skrip browser menjadi satu JavaScript file mandiri yang dapat Anda gunakan di browser.

Anda dapat membangun SDK sebagai dependensi pustaka untuk skrip browser apa pun dengan menggunakan Browserify. Misalnya, kode Node.js berikut memerlukan SDK:

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

Contoh kode ini dapat dikompilasi ke dalam versi yang kompatibel dengan browser menggunakan Browserify:

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

Aplikasi, termasuk dependensi SDK-nya, kemudian tersedia di browser melalui. `browser-app.js`

Untuk informasi selengkapnya tentang Browserify, lihat situs web Browserify.

Berbagi Sumber Daya Lintas Orisinil (CORS)

Berbagi sumber daya lintas asal, atau CORS, adalah fitur keamanan browser web modern. Ini memungkinkan browser web untuk menegosiasikan domain mana yang dapat membuat permintaan situs web atau layanan eksternal. CORS adalah pertimbangan penting ketika mengembangkan aplikasi browser dengan AWS SDK untuk JavaScript karena sebagian besar permintaan ke sumber daya dikirim ke domain eksternal, seperti titik akhir untuk layanan web. Jika JavaScript lingkungan Anda memberlakukan keamanan CORS, Anda harus mengonfigurasi CORS dengan layanan.

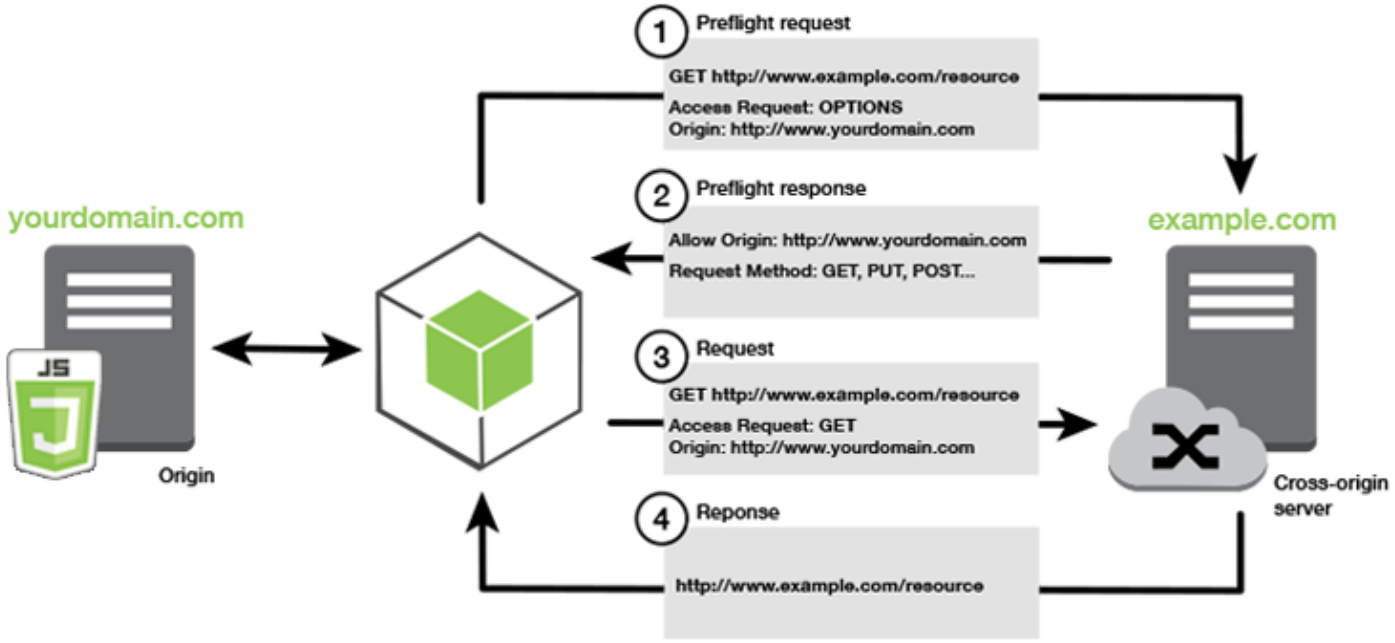
CORS menentukan apakah akan mengizinkan pembagian sumber daya dalam permintaan lintas asal berdasarkan:

- Domain spesifik yang membuat permintaan
- Jenis permintaan HTTP yang dibuat (GET, PUT, POST, DELETE dan sebagainya)

Bagaimana CORS Bekerja

Dalam kasus yang paling sederhana, skrip browser Anda membuat permintaan GET untuk sumber daya dari server di domain lain. Bergantung pada konfigurasi CORS server tersebut, jika permintaan berasal dari domain yang berwenang untuk mengirimkan permintaan GET, server lintas asal merespons dengan mengembalikan sumber daya yang diminta.

Jika domain yang meminta atau jenis permintaan HTTP tidak diotorisasi, permintaan ditolak. Namun, CORS memungkinkan untuk melakukan pra-penerbangan permintaan sebelum benar-benar mengirimkannya. Dalam hal ini, permintaan preflight dibuat di mana operasi permintaan `OPTIONS` akses dikirim. Jika konfigurasi CORS server lintas asal memberikan akses ke domain yang meminta, server mengirimkan kembali respons preflight yang mencantumkan semua jenis permintaan HTTP yang dapat dibuat oleh domain permintaan pada sumber daya yang diminta.



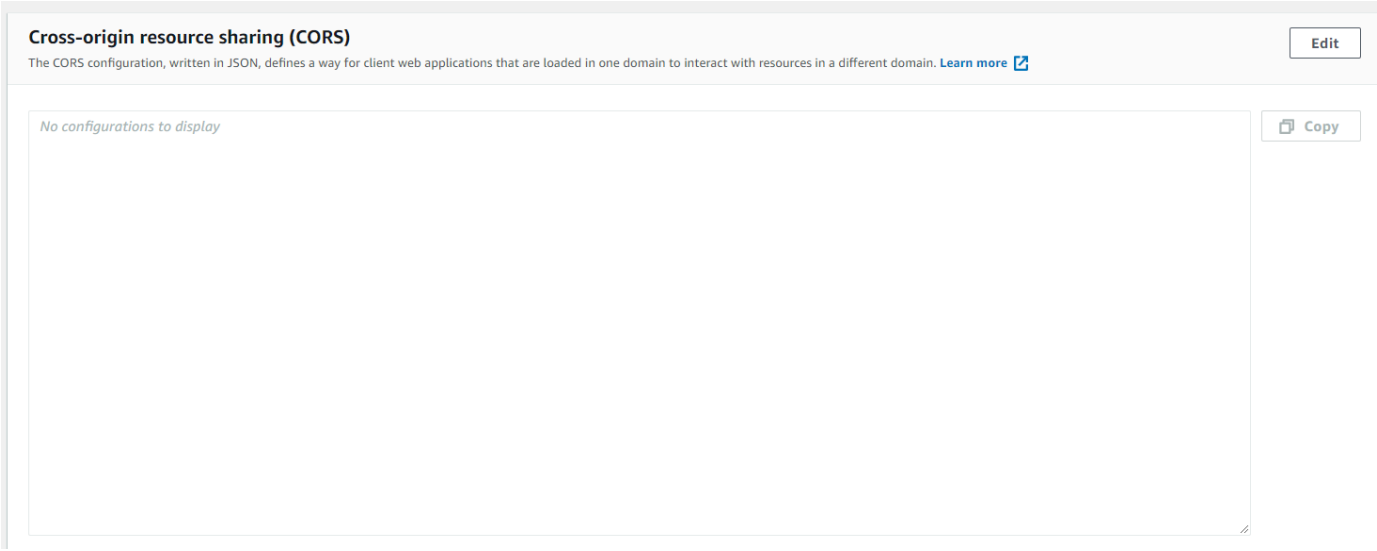
Apakah Konfigurasi CORS Diperlukan

Bucket Amazon S3 memerlukan konfigurasi CORS sebelum Anda dapat melakukan operasi pada mereka. Di beberapa JavaScript lingkungan CORS mungkin tidak diberlakukan dan oleh karena itu mengonfigurasi CORS tidak diperlukan. Misalnya, jika Anda meng-host aplikasi dari bucket Amazon S3 dan mengakses sumber daya dari `*.s3.amazonaws.com` atau titik akhir tertentu lainnya, permintaan Anda tidak akan mengakses domain eksternal. Oleh karena itu, konfigurasi ini tidak memerlukan CORS. Dalam hal ini, CORS masih digunakan untuk layanan selain Amazon S3.

Mengonfigurasi CORS untuk Bucket Amazon S3

Anda dapat mengonfigurasi bucket Amazon S3 untuk menggunakan CORS di konsol Amazon S3.

1. Di konsol Amazon S3, pilih bucket yang ingin Anda edit.
2. Pilih tab Izin, dan tekan ke panel Cross-Origin Resource Sharing (CORS).



3. Pilih Edit, dan ketik konfigurasi CORS Anda di CORS Configuration Editor, lalu pilih Simpan.

Konfigurasi CORS adalah file XMLyang berisi serangkaian aturan dalam file. `<CORSRule>` Konfigurasi dapat memiliki hingga 100 aturan. Aturan didefinisikan oleh salah satu tag berikut:

- `<AllowedOrigin>`, yang menentukan asal domain yang Anda izinkan untuk membuat permintaan lintas domain.
- `<AllowedMethod>`, yang menentukan jenis permintaan yang Anda izinkan (GET, PUT, POST, DELETE, HEAD) dalam permintaan lintas domain.
- `<AllowedHeader>`, yang menentukan header yang diizinkan dalam permintaan preflight.

Untuk contoh konfigurasi, lihat Bagaimana Cara Mengonfigurasi CORS di Bucket Saya? di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Contoh Konfigurasi CORS

Contoh konfigurasi CORS berikut memungkinkan pengguna untuk melihat, menambah, menghapus, atau memperbarui objek di dalam bucket dari domain`example.org`, meskipun Anda disarankan untuk memasukkan `<AllowedOrigin>` ke domain situs web Anda. Anda dapat menentukan `"*"` untuk mengizinkan asal apa pun.

> **Important**
>
> Pada konsol S3 baru, konfigurasi CORS harus 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"]
    }
]
```

Konfigurasi ini tidak mengizinkan pengguna untuk melakukan tindakan pada bucket. Ini memungkinkan model keamanan browser untuk memungkinkan permintaan ke Amazon S3. Izin harus dikonfigurasi melalui izin bucket atau izin peran IAM.

Anda dapat menggunakan `ExposeHeader` untuk membiarkan SDK membaca header respons yang dikembalikan dari Amazon S3. Misalnya, jika Anda ingin membaca `ETag` header dari unggahan PUT atau multipart, Anda perlu menyertakan `ExposeHeader` tag dalam konfigurasi Anda, seperti yang ditunjukkan pada contoh sebelumnya. SDK hanya dapat mengakses header yang diekspos melalui konfigurasi CORS. Jika Anda mengatur metadata pada objek, nilai dikembalikan sebagai header dengan awalan`x-amz-meta-`, sepert`x-amz-meta-my-custom-header`, dan juga harus diekspos dengan cara yang sama.

Bundling Aplikasi dengan Webpack

Aplikasi web dalam skrip browser atau penggunaan modul kode Node.js menciptakan dependensi. Modul kode ini dapat memiliki dependensi sendiri, menghasilkan kumpulan modul yang saling berhubungan yang dibutuhkan aplikasi Anda untuk berfungsi. Untuk mengelola dependensi, Anda dapat menggunakan bundler modul seperti webpack.

Bundler modul webpack mengurai kode aplikasi Anda, mencari `import` atau `require` pernyataan, untuk membuat bundel yang berisi semua aset yang dibutuhkan aplikasi Anda sehingga aset dapat dengan mudah dilayani melalui halaman web. SDK for JavaScript dapat disertakan dalam webpack sebagai salah satu dependensi untuk disertakan dalam bundel keluaran.

Untuk informasi selengkapnya tentang webpack, lihat bundler modul webpack aktif. GitHub

Menginstal Webpack

Untuk menginstal bundler modul webpack, Anda harus menginstal npm, manajer paket Node.js terlebih dahulu. Ketik perintah berikut untuk menginstal CLI JavaScript dan modul webpack.

```
npm install webpack
```

Anda mungkin juga perlu menginstal plugin webpack yang memungkinkannya memuat file JSON. Ketik perintah berikut untuk menginstal plugin pemuat JSON.

```
npm install json-loader
```

Mengkonfigurasi Webpack

Secara default, webpack mencari JavaScript file bernama `webpack.config.js` di direktori root proyek Anda. File ini menentukan opsi konfigurasi Anda. Berikut adalah contoh file `webpack.config.js` konfigurasi.

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

Dalam contoh ini, `browser.js` ditentukan sebagai titik masuk. Titik masuknya adalah file yang digunakan webpack untuk mulai mencari modul yang diimpor. Nama file output ditentukan sebagai`bundle.js`. File output ini akan berisi semua aplikasi JavaScript yang perlu dijalankan. Jika kode yang ditentukan di titik masuk mengimpor atau memerlukan modul lain, seperti SDK for JavaScript, kode tersebut dibundel tanpa perlu menentukannya dalam konfigurasi.

Konfigurasi di `json-loader` plugin yang diinstal sebelumnya menentukan untuk webpack cara mengimpor file JSON. Secara default, webpack hanya mendukung JavaScript tetapi menggunakan

loader untuk menambahkan dukungan untuk mengimpor jenis file lain. Karena SDK untuk JavaScript menggunakan file JSON secara ekstensif, webpack menimbulkan kesalahan saat membuat bundel jika `json-loader` tidak disertakan.

Menjalankan Webpack

Untuk membangun aplikasi untuk menggunakan webpack, tambahkan berikut ini ke `scripts` objek dalam `package.json` file Anda.

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

Berikut adalah contoh `package.json` yang menunjukkan penambahan 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"
  }
}
```

Untuk membangun aplikasi Anda, ketik perintah berikut.

```
npm run build
```

Bundler modul webpack kemudian menghasilkan JavaScript file yang Anda tentukan di direktori root proyek Anda.

Menggunakan Webpack Bundle

Untuk menggunakan bundel dalam skrip browser, Anda dapat menggabungkan bundel menggunakan `<script>` tag seperti yang ditunjukkan pada contoh berikut.

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

Mengimpor Layanan Individu

Salah satu manfaat webpack adalah mem-parsing dependensi dalam kode Anda dan hanya menggabungkan kode yang dibutuhkan aplikasi Anda. Jika Anda menggunakan SDK untuk JavaScript, menggabungkan hanya bagian SDK yang benar-benar digunakan oleh aplikasi Anda dapat mengurangi ukuran output webpack secara signifikan.

Perhatikan contoh kode berikut yang digunakan untuk membuat objek layanan 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'});
```

`require()`Fungsi menentukan seluruh SDK. Bundel webpack yang dihasilkan dengan kode ini akan menyertakan SDK lengkap tetapi SDK lengkap tidak diperlukan jika hanya kelas klien Amazon S3 yang digunakan. Ukuran bundel akan jauh lebih kecil jika hanya bagian SDK yang Anda perlukan untuk layanan Amazon S3 yang disertakan. Bahkan menyetel konfigurasi tidak memerlukan SDK lengkap karena Anda dapat mengatur data konfigurasi pada objek layanan Amazon S3.

Inilah yang terlihat seperti kode yang sama ketika hanya menyertakan bagian Amazon S3 dari 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}
  });
```

Bundling untuk Node.js

Anda dapat menggunakan webpack untuk menghasilkan bundel yang berjalan di Node.js dengan menentukannya sebagai target dalam konfigurasi.

```
target: "node"
```

Ini berguna saat menjalankan aplikasi Node.js di lingkungan di mana ruang disk terbatas. Berikut adalah contoh `webpack.config.js` konfigurasi dengan Node.js ditentukan sebagai target output.

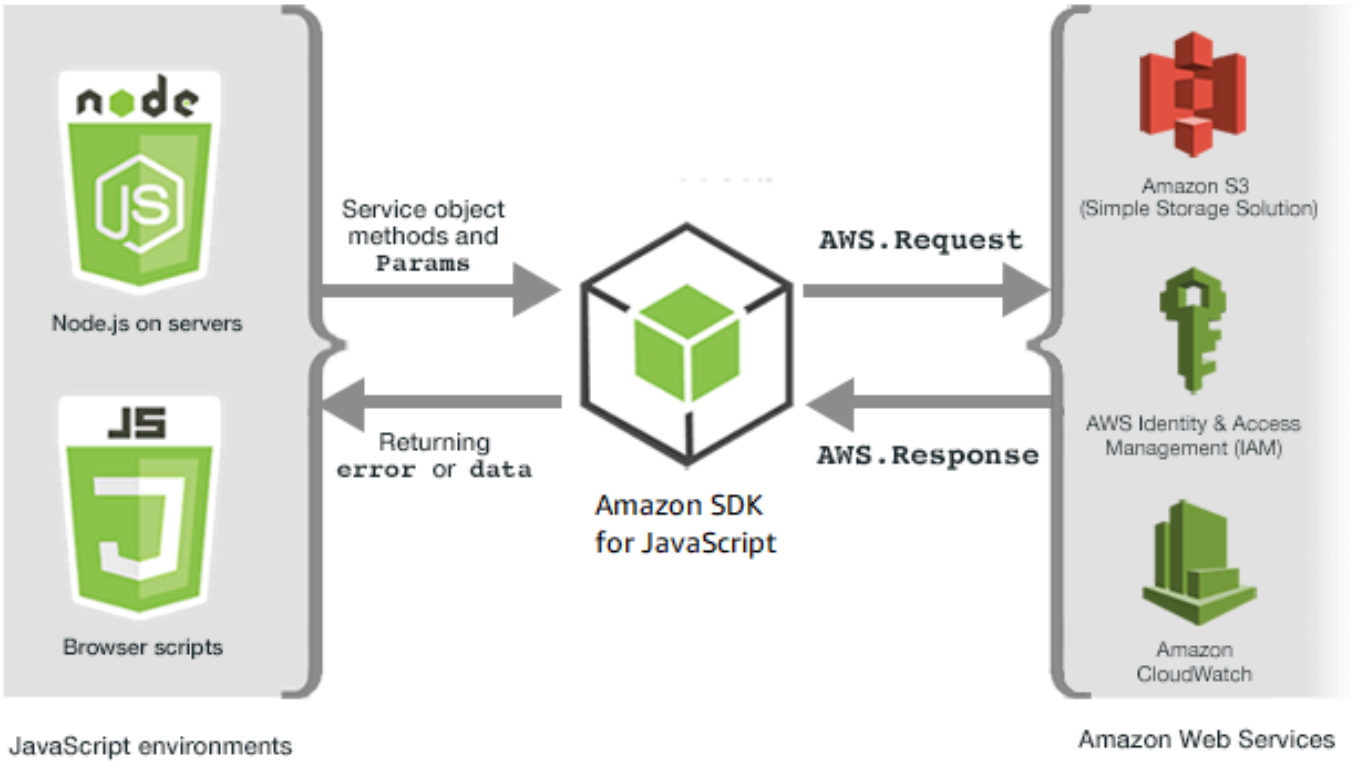
```
// Import path for resolving file paths
var path = require('path');
module.exports = {
  // Specify the entry point for our app
  entry: [
    path.join(__dirname, 'node.js')
  ],
  // Specify the output file containing our bundled code
  output: {
    path: __dirname,
    filename: 'bundle.js'
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

Bekerja dengan Layanan di SDK untuk JavaScript

AWS SDK untuk JavaScript Menyediakan akses ke layanan yang didukungnya melalui kumpulan kelas klien. Dari kelas klien ini, Anda membuat objek antarmuka layanan, yang biasa disebut objek layanan. Setiap AWS layanan yang didukung memiliki satu atau lebih kelas klien yang menawarkan tingkat rendah APIs untuk menggunakan fitur dan sumber daya layanan. Misalnya, Amazon APIs DynamoDB tersedia melalui kelas. `AWS.DynamoDB`

Layanan yang diekspos melalui SDK untuk JavaScript mengikuti pola permintaan-respons untuk bertukar pesan dengan aplikasi panggilan. Dalam pola ini, kode yang menjalankan layanan mengirimkan permintaan HTTP/HTTPS ke titik akhir untuk layanan. Permintaan berisi parameter yang diperlukan untuk berhasil memanggil fitur tertentu yang dipanggil. Layanan yang dipanggil menghasilkan respons yang dikirim kembali ke pemohon. Respons berisi data jika operasi berhasil atau informasi kesalahan jika operasi tidak berhasil.



Memanggil AWS layanan mencakup siklus hidup permintaan dan respons penuh dari operasi pada objek layanan, termasuk percobaan ulang apa pun yang dicoba. Permintaan dienkapsulasi dalam SDK oleh objek. `AWS.Request` Respons dienkapsulasi dalam SDK oleh `AWS.Response` objek, yang diberikan kepada pemohon melalui salah satu dari beberapa teknik, seperti fungsi panggilan balik atau janji. JavaScript

Topik

- Membuat dan Memanggil Objek Layanan
- AWS SDK untuk JavaScript Panggilan Pencatatan
- Layanan Panggilan Secara Asinkron
- Menggunakan Response Object
- Bekerja dengan JSON
- Coba lagi strategi di v2 AWS SDK untuk JavaScript

Membuat dan Memanggil Objek Layanan

JavaScript API mendukung sebagian besar AWS layanan yang tersedia. Setiap kelas layanan di JavaScript API menyediakan akses ke setiap panggilan API dalam layanannya. Untuk informasi selengkapnya tentang kelas layanan, operasi, dan parameter di JavaScript API, lihat referensi API.

Saat menggunakan SDK di Node.js, Anda menambahkan paket SDK ke aplikasi Anda menggunakan`require`, yang menyediakan dukungan untuk semua layanan saat ini.

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

Saat menggunakan SDK dengan browser JavaScript, Anda memuat paket SDK ke skrip browser menggunakan paket SDK yang dihosting AWS. Untuk memuat paket SDK, tambahkan `<script>` elemen berikut:

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

Untuk menemukan SDK_VERSION_NUMBER saat ini, lihat Referensi API untuk SDK untuk Panduan Referensi API. JavaScript AWS SDK untuk JavaScript

Paket SDK yang di-host default menyediakan dukungan untuk subset dari layanan yang tersedia AWS . Untuk daftar layanan default dalam paket SDK yang dihosting untuk browser, lihat Layanan yang Didukung di Referensi API. Anda dapat menggunakan SDK dengan layanan lain jika pemeriksaan keamanan CORS dinonaktifkan. Dalam hal ini, Anda dapat membuat versi kustom SDK untuk menyertakan layanan tambahan yang Anda butuhkan. Untuk informasi selengkapnya tentang membuat versi kustom SDK, lihatMembangun SDK untuk Browser.

Membutuhkan Layanan Individu

Memerlukan SDK untuk JavaScript seperti yang ditunjukkan sebelumnya menyertakan seluruh SDK ke dalam kode Anda. Sebagai alternatif, Anda dapat memilih untuk hanya meminta layanan individual yang digunakan oleh kode Anda. Pertimbangkan kode berikut yang digunakan untuk membuat objek layanan 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'});
```

Pada contoh sebelumnya, `require` fungsi menentukan seluruh SDK. Jumlah kode untuk diangkut melalui jaringan serta overhead memori kode Anda akan jauh lebih kecil jika hanya bagian SDK yang Anda butuhkan untuk layanan Amazon S3 yang disertakan. Untuk memerlukan layanan individual, panggil `require` fungsi seperti yang ditunjukkan, termasuk konstruktor layanan dalam semua huruf kecil.

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

Berikut adalah kode untuk membuat objek layanan Amazon S3 sebelumnya ketika hanya menyertakan bagian Amazon S3 dari 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}
  });
```

Anda masih dapat mengakses AWS namespace global tanpa setiap layanan yang melekat padanya.

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

Ini adalah teknik yang berguna ketika menerapkan konfigurasi yang sama di beberapa layanan individual, misalnya untuk memberikan kredensyal yang sama untuk semua layanan. Membutuhkan layanan individual harus mengurangi waktu pemuatan dan konsumsi memori di Node.js. Ketika dilakukan bersama dengan alat bundling seperti Browserify atau webpack, memerlukan layanan individual menghasilkan SDK menjadi sebagian kecil dari ukuran penuh. Ini membantu dengan memori atau lingkungan terbatas ruang disk seperti perangkat IoT atau dalam fungsi Lambda.

Membuat Objek Layanan

Untuk mengakses fitur layanan melalui JavaScript API, pertama-tama Anda membuat objek layanan yang digunakan untuk mengakses serangkaian fitur yang disediakan oleh kelas klien yang mendasarinya. Umumnya ada satu kelas klien yang disediakan untuk setiap layanan; Namun, beberapa layanan membagi akses ke fitur mereka di antara beberapa kelas klien.

Untuk menggunakan fitur, Anda harus membuat instance dari kelas yang menyediakan akses ke fitur itu. Contoh berikut menunjukkan membuat objek layanan untuk DynamoDB dari `AWS.DynamoDB` kelas klien.

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

Secara default, objek layanan dikonfigurasi dengan pengaturan global yang juga digunakan untuk mengkonfigurasi SDK. Namun, Anda dapat mengonfigurasi objek layanan dengan data konfigurasi runtime yang spesifik untuk objek layanan tersebut. Data konfigurasi khusus layanan diterapkan setelah menerapkan pengaturan konfigurasi global.

Dalam contoh berikut, objek EC2 layanan Amazon dibuat dengan konfigurasi untuk Wilayah tertentu tetapi sebaliknya menggunakan konfigurasi global.

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

Selain mendukung konfigurasi khusus layanan yang diterapkan ke objek layanan individual, Anda juga dapat menerapkan konfigurasi khusus layanan ke semua objek layanan yang baru dibuat dari kelas tertentu. Misalnya, untuk mengonfigurasi semua objek layanan yang dibuat dari EC2 kelas Amazon untuk menggunakan Wilayah AS Barat (Oregon`us-west-2`) (), tambahkan berikut ini ke objek konfigurasi `AWS.config` global.

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

Mengunci Versi API dari Objek Layanan

Anda dapat mengunci objek layanan ke versi API tertentu dari layanan dengan menentukan `apiVersion` opsi saat membuat objek. Dalam contoh berikut, objek layanan DynamoDB dibuat yang dikunci ke versi API tertentu.

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

Untuk informasi selengkapnya tentang mengunci versi API dari objek layanan, lihatMengunci Versi API.

Menentukan Parameter Objek Layanan

Saat memanggil metode objek layanan, berikan parameter di JSON seperti yang dipersyaratkan oleh API. Misalnya, di Amazon S3, untuk mendapatkan objek untuk bucket dan kunci tertentu, teruskan parameter berikut ke metode. `getObject` Untuk informasi selengkapnya tentang meneruskan parameter JSON, lihatBekerja dengan JSON.

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

Untuk informasi selengkapnya tentang parameter Amazon S3, lihat `Class: AWS.S3`di referensi API.

Selain itu, Anda dapat mengikat nilai ke parameter individual saat membuat objek layanan menggunakan `params` parameter. Nilai `params` parameter objek layanan adalah peta yang menentukan satu atau lebih nilai parameter yang ditentukan oleh objek layanan. Contoh berikut menunjukkan `Bucket` parameter objek layanan Amazon S3 yang terikat ke bucket bernama. `amzn-s3-demo-bucket`

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

Dengan mengikat objek layanan ke bucket, objek `s3bucket` layanan memperlakukan nilai `amzn-s3-demo-bucket` parameter sebagai nilai default yang tidak lagi perlu ditentukan untuk operasi selanjutnya. Setiap nilai parameter terikat diabaikan saat menggunakan objek untuk operasi di mana nilai parameter tidak berlaku. Anda dapat mengganti parameter terikat ini saat melakukan panggilan pada objek layanan dengan menentukan nilai baru.

```
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'});
```

Detail tentang parameter yang tersedia untuk setiap metode dapat ditemukan di referensi API.

AWS SDK untuk JavaScript Panggilan Pencatatan

AWS SDK untuk JavaScript Ini diinstrumentasi dengan logger bawaan sehingga Anda dapat mencatat panggilan API yang Anda buat dengan SDK untuk. JavaScript

Untuk mengaktifkan logger dan mencetak entri log di konsol, tambahkan pernyataan berikut ke kode Anda.

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

Berikut adalah contoh dari output log.

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

Menggunakan Logger Pihak Ketiga

Anda juga dapat menggunakan logger pihak ketiga, asalkan memiliki `log()` atau `write()` operasi untuk menulis ke file log atau server. Anda harus menginstal dan mengatur logger kustom Anda seperti yang diinstruksikan sebelum Anda dapat menggunakannya dengan SDK for. JavaScript

Salah satu logger yang dapat Anda gunakan di skrip browser atau di Node.js adalah logplease. Di Node.js, Anda dapat mengkonfigurasi logplease untuk menulis entri log ke file log. Anda juga dapat menggunakannya dengan webpack.

Saat menggunakan logger pihak ketiga, setel semua opsi sebelum menetapkan logger ke. `AWS.Config.logger` Misalnya, berikut ini menentukan file log eksternal dan menetapkan tingkat log untuk 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;
```

Untuk informasi lebih lanjut tentang logplease, lihat logplease Simple JavaScript Logger on. GitHub

Layanan Panggilan Secara Asinkron

Semua permintaan yang dibuat melalui SDK bersifat asinkron. Ini penting untuk diingat saat menulis skrip browser. JavaScript berjalan di browser web biasanya hanya memiliki satu thread eksekusi. Setelah melakukan panggilan asinkron ke AWS layanan, skrip browser terus berjalan dan dalam proses dapat mencoba mengeksekusi kode yang bergantung pada hasil asinkron sebelum kembali.

Membuat panggilan asinkron ke AWS layanan termasuk mengelola panggilan tersebut sehingga kode Anda tidak mencoba menggunakan data sebelum data tersedia. Topik di bagian ini menjelaskan perlunya mengelola panggilan asinkron dan merinci berbagai teknik yang dapat Anda gunakan untuk mengelolanya.

Topik

- Mengelola Panggilan Asinkron
- Menggunakan Fungsi Callback Anonim
- Menggunakan Request Object Event Listener
- Menggunakan async/await
- Menggunakan JavaScript Janji

Mengelola Panggilan Asinkron

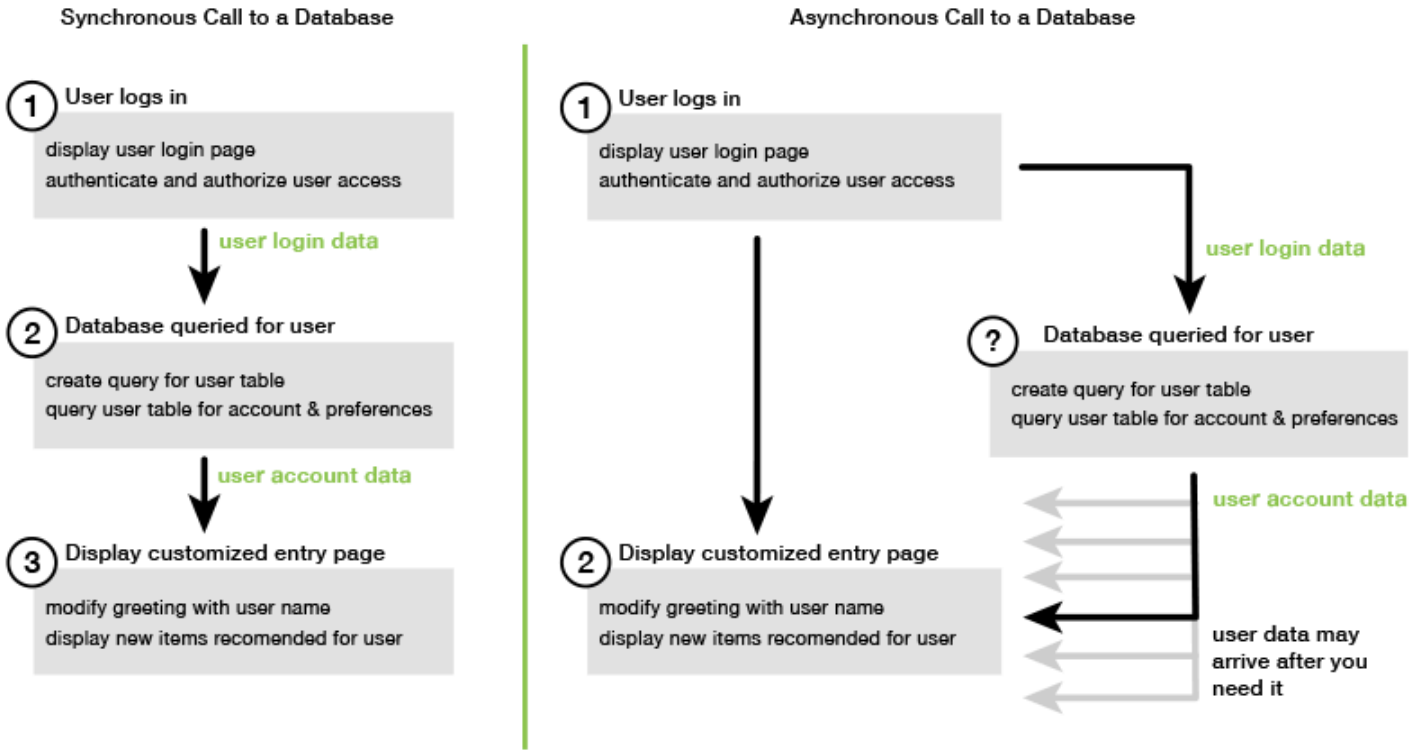
Misalnya, halaman beranda situs web e-commerce memungkinkan pelanggan yang kembali masuk. Bagian dari manfaat bagi pelanggan yang masuk adalah bahwa, setelah masuk, situs kemudian menyesuaikan diri dengan preferensi khusus mereka. Untuk membuat ini terjadi:

1. Pelanggan harus masuk dan divalidasi dengan kredensi masuk mereka.
2. Preferensi pelanggan diminta dari database pelanggan.

3. Basis data menyediakan preferensi pelanggan yang digunakan untuk menyesuaikan situs sebelum halaman dimuat.

Jika tugas-tugas ini dijalankan secara serempak, maka masing-masing harus selesai sebelum tugas berikutnya dapat dimulai. Halaman web tidak akan dapat menyelesaikan pemuatan sampai preferensi pelanggan kembali dari database. Namun, setelah kueri database dikirim ke server, penerimaan data pelanggan dapat ditunda atau bahkan gagal karena kemacetan jaringan, lalu lintas basis data yang sangat tinggi, atau koneksi perangkat seluler yang buruk.

Agar situs web tidak membeku dalam kondisi tersebut, hubungi database secara tidak sinkron. Setelah panggilan database dijalankan, mengirimkan permintaan asinkron Anda, kode Anda terus dijalankan seperti yang diharapkan. Jika Anda tidak mengelola respons panggilan asinkron dengan benar, kode Anda dapat mencoba menggunakan informasi yang diharapkan kembali dari database ketika data tersebut belum tersedia.



Menggunakan Fungsi Callback Anonim

Setiap metode objek layanan yang membuat `AWS.Request` objek dapat menerima fungsi callback anonim sebagai parameter terakhir. Tanda tangan dari fungsi callback ini adalah:

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

```
    // callback handling code
}
```

Fungsi callback ini dijalankan ketika respon berhasil atau data kesalahan kembali. Jika pemanggilan metode berhasil, isi respons tersedia untuk fungsi callback dalam parameter. `data` Jika panggilan tidak berhasil, detail tentang kegagalan disediakan dalam `error` parameter.

Biasanya kode di dalam fungsi callback menguji kesalahan, yang diproses jika dikembalikan. Jika kesalahan tidak dikembalikan, kode kemudian mengambil data dalam respons dari `data` parameter. Bentuk dasar dari fungsi callback terlihat seperti contoh ini.

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

Pada contoh sebelumnya, detail kesalahan atau data yang dikembalikan dicatat ke konsol. Berikut adalah contoh yang menunjukkan fungsi callback diteruskan sebagai bagian dari memanggil metode pada objek layanan.

```
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
  }
});
```

Mengakses Objek Permintaan dan Respons

Dalam fungsi callback, JavaScript kata kunci `this` mengacu pada `AWS.Response` objek yang mendasari untuk sebagian besar layanan. Dalam contoh berikut, `httpResponse` properti `AWS.Response` objek digunakan dalam fungsi callback untuk mencatat data respons mentah dan header untuk membantu debugging.

```
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
  }
});
```

Selain itu, karena `AWS.Response` objek memiliki `Request` properti yang berisi `AWS.Request` yang dikirim oleh panggilan metode asli, Anda juga dapat mengakses detail permintaan yang dibuat.

Menggunakan Request Object Event Listener

Jika Anda tidak membuat dan meneruskan fungsi callback anonim sebagai parameter saat Anda memanggil metode objek layanan, panggilan metode menghasilkan `AWS.Request` objek yang harus dikirim secara manual menggunakan `send` metodenya.

Untuk memproses respons, Anda harus membuat event listener untuk `AWS.Request` objek untuk mendaftarkan fungsi callback untuk pemanggilan metode. Contoh berikut menunjukkan cara membuat `AWS.Request` objek untuk memanggil metode objek layanan dan pendengar acara untuk pengembalian yang berhasil.

```
// 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();
```

Setelah `send` metode pada `AWS.Request` objek dipanggil, event handler mengeksekusi ketika objek layanan menerima objek`AWS.Response`.

Untuk informasi selengkapnya tentang `AWS.Request` objek, lihat `Class: AWS.Request`di Referensi API. Untuk informasi selengkapnya tentang `AWS.Response` objek, lihat Menggunakan Response Object atau `Class: AWS.Response`di Referensi API.

Merantai Beberapa Panggilan Balik

Anda dapat mendaftarkan beberapa callback pada objek permintaan apa pun. Beberapa panggilan balik dapat didaftarkan untuk acara yang berbeda atau acara yang sama. Selain itu, Anda dapat menghubungkan callback seperti yang ditunjukkan pada contoh berikut.

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

Permintaan Acara Penyelesaian Objek

`AWS.Request`Objek memunculkan peristiwa penyelesaian ini berdasarkan respons dari setiap metode operasi layanan:

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

Anda dapat mendaftarkan fungsi callback dalam menanggapi salah satu peristiwa ini. Untuk daftar lengkap semua peristiwa objek permintaan, lihat `Class: AWS.Request`di Referensi API.

Acara Sukses

`success`Acara ini diangkat setelah respons yang berhasil diterima dari objek layanan. Berikut adalah cara Anda mendaftarkan fungsi callback untuk acara ini.

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

Respons menyediakan `data` properti yang berisi data respons serial dari layanan. Misalnya, panggilan berikut ke `listBuckets` metode objek layanan Amazon S3

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

mengembalikan respon dan kemudian mencetak konten `data` properti berikut ke konsol.

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

Peristiwa kesalahan

`error`Acara ini muncul pada respons kesalahan yang diterima dari objek layanan. Berikut adalah cara Anda mendaftarkan fungsi callback untuk acara ini.

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

Ketika `error` acara dinaikkan, nilai `data` properti respon adalah `null` dan `error` properti berisi data kesalahan. `error`Objek terkait diteruskan sebagai parameter pertama ke fungsi callback terdaftar. Misalnya, kode berikut:

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

mengembalikan kesalahan dan kemudian mencetak data kesalahan berikut ke konsol.

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

Acara Lengkap

`complete`Acara dimunculkan ketika panggilan objek layanan telah selesai, terlepas dari apakah panggilan tersebut menghasilkan keberhasilan atau kesalahan. Berikut adalah cara Anda mendaftarkan fungsi callback untuk acara ini.

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

Gunakan callback `complete` acara untuk menangani pembersihan permintaan apa pun yang harus dijalankan terlepas dari keberhasilan atau kesalahan. Jika Anda menggunakan data respons di dalam callback untuk `complete` acara tersebut, periksa dulu `response.error` properti `response.data` atau sebelum mencoba mengakses salah satunya, seperti yang ditunjukkan pada contoh berikut.

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

Permintaan Objek Acara HTTP

`AWS.Request`Objek memunculkan peristiwa HTTP ini berdasarkan respons dari setiap metode operasi layanan:

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

Anda dapat mendaftarkan fungsi callback dalam menanggapi salah satu peristiwa ini. Untuk daftar lengkap semua peristiwa objek permintaan, lihat `Class: AWS.Request`di Referensi API.

Acara HttpHeaders

`httpHeaders`Acara ini dinaikkan ketika header dikirim oleh server jarak jauh. Berikut adalah cara Anda mendaftarkan fungsi callback untuk acara ini.

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

`statusCode`Parameter untuk fungsi callback adalah kode status HTTP. `headers`Parameter berisi header respons.

Acara HttpData

`httpData`Acara ini dinaikkan untuk mengalirkan paket data respons dari layanan. Berikut adalah cara Anda mendaftarkan fungsi callback untuk acara ini.

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

Acara ini biasanya digunakan untuk menerima respons besar dalam potongan saat memuat seluruh respons ke dalam memori tidak praktis. Acara ini memiliki `chunk` parameter tambahan yang berisi sebagian data aktual dari server.

Jika Anda mendaftarkan callback untuk `httpData` acara tersebut, `data` properti respons berisi seluruh keluaran serial untuk permintaan tersebut. Anda harus menghapus `httpData` pendengar default jika Anda tidak memiliki parsing tambahan dan overhead memori untuk penangan bawaan.

httpUploadProgress dan httpDownloadProgress Peristiwa

`httpUploadProgress`Acara ini muncul ketika permintaan HTTP telah mengunggah lebih banyak data. Demikian pula, `httpDownloadProgress` acara dinaikkan ketika permintaan HTTP telah mengunduh lebih banyak data. Berikut adalah cara Anda mendaftarkan fungsi callback untuk acara ini.

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

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

`progress`Parameter ke fungsi callback berisi objek dengan byte yang dimuat dan total permintaan.

Acara HttpError

`httpError`Acara dimunculkan ketika permintaan HTTP gagal. Berikut adalah cara Anda mendaftarkan fungsi callback untuk acara ini.

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

`error`Parameter ke fungsi callback berisi kesalahan yang dilemparkan.

Acara HttpDone

`httpDone`Acara ini dinaikkan ketika server selesai mengirim data. Berikut adalah cara Anda mendaftarkan fungsi callback untuk acara ini.

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

Menggunakan async/await

Anda dapat menggunakan `async/await` pola dalam panggilan Anda ke file AWS SDK untuk JavaScript. Sebagian besar fungsi yang mengambil panggilan balik tidak mengembalikan janji. Karena Anda hanya menggunakan `await` fungsi yang mengembalikan janji, untuk menggunakan `async/await` pola Anda perlu menghubungkan `.promise()` metode ke akhir panggilan Anda, dan menghapus panggilan balik.

Contoh berikut menggunakan async/await untuk mencantumkan semua tabel Amazon DynamoDB Anda. `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**
>
> Tidak semua browser mendukung async/await. Lihat Fungsi async untuk daftar browser dengan dukungan async/await.

Menggunakan JavaScript Janji

`AWS.Request.promise`Metode ini menyediakan cara untuk memanggil operasi layanan dan mengelola aliran asinkron alih-alih menggunakan callback. Dalam Node.js dan skrip browser, `AWS.Request` objek dikembalikan ketika operasi layanan dipanggil tanpa fungsi callback. Anda dapat memanggil `send` metode permintaan untuk melakukan panggilan layanan.

Namun, `AWS.Request.promise` segera mulai panggilan layanan dan mengembalikan janji yang dipenuhi dengan `data` properti respons atau ditolak dengan `error` properti respons.

```
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 */
  }
```

```
);
```

Contoh berikutnya mengembalikan janji yang dipenuhi dengan `data` objek, atau ditolak dengan `error` objek. Menggunakan janji, satu panggilan balik tidak bertanggung jawab untuk mendeteksi kesalahan. Sebaliknya, callback yang benar dipanggil berdasarkan keberhasilan atau kegagalan permintaan.

```
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);
});
```

Koordinasi Beberapa Janji

Dalam beberapa situasi, kode Anda harus membuat beberapa panggilan asinkron yang memerlukan tindakan hanya jika semuanya berhasil dikembalikan. Jika Anda mengelola panggilan metode asinkron individual tersebut dengan janji, Anda dapat membuat janji tambahan yang menggunakan metode ini. `all` Metode ini memenuhi janji payung ini jika dan ketika berbagai janji yang Anda berikan ke metode terpenuhi. Fungsi callback dilewatkan sebuah array dari nilai-nilai dari janji-janji yang diteruskan ke `all` metode.

Dalam contoh berikut, AWS Lambda fungsi harus membuat tiga panggilan asinkron ke Amazon DynamoDB tetapi hanya dapat diselesaikan setelah janji untuk setiap panggilan terpenuhi.

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

  console.log("Value 0 is " + values[0].toString);
  console.log("Value 1 is " + values[1].toString);
  console.log("Value 2 is " + values[2].toString);

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

Browser dan Node.js Support for Promises

Support for native JavaScript promises (ECMAScript 2015) tergantung pada JavaScript mesin dan versi di mana kode Anda dijalankan. Untuk membantu menentukan dukungan untuk JavaScript janji di setiap lingkungan tempat kode Anda perlu dijalankan, lihat Tabel ECMAScript Kompatibilitas di. GitHub

Menggunakan Implementasi Janji Lainnya

Selain implementasi janji asli pada tahun ECMAScript 2015, Anda juga dapat menggunakan pustaka janji pihak ketiga, termasuk:

- burung biru
- RSVP
- Q

Pustaka janji opsional ini dapat berguna jika Anda memerlukan kode Anda untuk berjalan di lingkungan yang tidak mendukung implementasi janji asli di ECMAScript 5 dan ECMAScript 2015.

Untuk menggunakan pustaka janji pihak ketiga, tetapkan dependensi janji pada SDK dengan memanggil `setPromisesDependency` metode objek konfigurasi global. Dalam skrip browser, pastikan untuk memuat pustaka janji pihak ketiga sebelum memuat SDK. Dalam contoh berikut, SDK dikonfigurasi untuk menggunakan implementasi di pustaka janji bluebird.

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

Untuk kembali menggunakan implementasi janji asli JavaScript mesin, panggil `setPromisesDependency` lagi, berikan nama `null` alih-alih perpustakaan.

Menggunakan Response Object

Setelah metode objek layanan dipanggil, ia mengembalikan `AWS.Response` objek dengan meneruskannya ke fungsi callback Anda. Anda mengakses konten respons melalui properti `AWS.Response` objek. Ada dua properti `AWS.Response` objek yang Anda gunakan untuk mengakses isi respons:

- `data`properti

- `error`properti

Saat menggunakan mekanisme callback standar, kedua properti ini disediakan sebagai parameter pada fungsi callback anonim seperti yang ditunjukkan pada contoh berikut.

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

Mengakses Data yang Dikembalikan di Objek Response

`data`Properti `AWS.Response` objek berisi data serial yang dikembalikan oleh permintaan layanan. Ketika permintaan berhasil, `data` properti berisi objek yang berisi peta ke data yang dikembalikan. `data`Properti bisa null jika terjadi kesalahan.

Berikut adalah contoh memanggil `getItem` metode tabel DynamoDB untuk mengambil nama file file gambar untuk digunakan sebagai bagian dari permainan.

```
// 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();
```

Untuk contoh ini, tabel DynamoDB adalah pencarian gambar yang menunjukkan hasil tarikan mesin slot seperti yang ditentukan oleh parameter di. `slotParams`

Setelah pemanggilan `getItem` metode berhasil, `data` properti `AWS.Response` objek berisi `Item` objek yang dikembalikan oleh DynamoDB. Data yang dikembalikan diakses sesuai dengan `ProjectionExpression` parameter permintaan, yang dalam hal ini berarti `imageFile` anggota `Item` objek. Karena `imageFile` anggota memegang nilai string, Anda mengakses nama file dari gambar itu sendiri melalui nilai anggota `S` anak dari`imageFile`.

Paging Melalui Data yang Dikembalikan

Terkadang konten `data` properti yang dikembalikan oleh permintaan layanan menjangkau beberapa halaman. Anda dapat mengakses halaman data berikutnya dengan memanggil `response.nextPage` metode. Metode ini mengirimkan permintaan baru. Respons dari permintaan dapat ditangkap baik dengan callback atau dengan sukses dan pendengar kesalahan.

Anda dapat memeriksa untuk melihat apakah data yang dikembalikan oleh permintaan layanan memiliki halaman data tambahan dengan memanggil `response.hasNextPage` metode. Metode ini mengembalikan boolean untuk menunjukkan apakah panggilan `response.nextPage` mengembalikan data tambahan.

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

Mengakses Informasi Kesalahan dari Objek Respons

`error`Properti `AWS.Response` objek berisi data kesalahan yang tersedia jika terjadi kesalahan layanan atau kesalahan transfer. Kesalahan yang dikembalikan mengambil formulir berikut.

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

Dalam kasus kesalahan, nilai `data` properti adalah`null`. Jika Anda menangani peristiwa yang dapat berada dalam status kegagalan, selalu periksa apakah `error` properti telah disetel sebelum mencoba mengakses nilai `data` properti.

Mengakses Objek Permintaan Asal

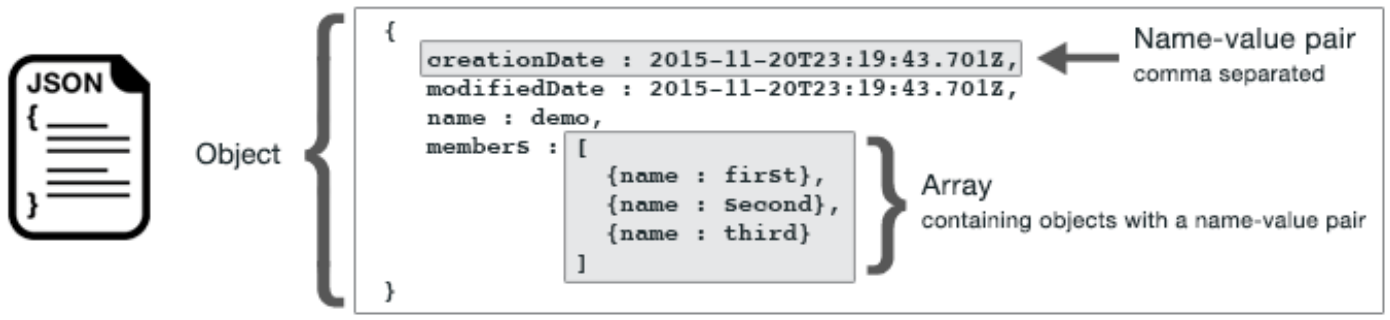
`request`Properti menyediakan akses ke `AWS.Request` objek asal. Hal ini dapat berguna untuk merujuk ke `AWS.Request` objek asli untuk mengakses parameter asli yang dikirim. Dalam contoh berikut, `request` properti digunakan untuk mengakses `Key` parameter permintaan layanan asli.

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

Bekerja dengan JSON

JSON adalah format untuk pertukaran data yang dapat dibaca manusia dan mesin. Sementara nama JSON adalah akronim dari JavaScript Object Notation, format JSON tidak tergantung pada bahasa pemrograman apa pun.

SDK untuk JavaScript menggunakan JSON untuk mengirim data ke objek layanan saat membuat permintaan dan menerima data dari objek layanan sebagai JSON. Untuk informasi lebih lanjut tentang JSON, lihat json.org.



JSON mewakili data dalam dua cara:

- Objek, yang merupakan kumpulan pasangan nilai nama yang tidak dipesan. Sebuah objek didefinisikan dalam kurung kurung kiri ({) dan kanan (}). Setiap pasangan nilai dimulai dengan nama, diikuti dengan titik dua, diikuti dengan nilai. Pasangan nama-nilai dipisahkan koma.
- Sebuah rangkaian, yang merupakan kumpulan nilai yang dipesan. Array didefinisikan dalam tanda kurung kiri ([) dan kanan (]). Item dalam array dipisahkan koma.

Berikut adalah contoh objek JSON yang berisi array objek di mana objek mewakili kartu dalam permainan kartu. Setiap kartu didefinisikan oleh dua pasangan nama-nilai, satu yang menentukan

nilai unik untuk mengidentifikasi kartu itu dan satu lagi yang menentukan URL yang menunjuk ke gambar kartu yang sesuai.

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

JSON sebagai Parameter Objek Layanan

Berikut adalah contoh JSON sederhana yang digunakan untuk menentukan parameter panggilan ke objek layanan Lambda.

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

`pullParams`Objek didefinisikan oleh tiga pasangan nama-nilai, dipisahkan oleh koma di dalam kurung kiri dan kanan. Saat memberikan parameter ke panggilan metode objek layanan, nama ditentukan oleh nama parameter untuk metode objek layanan yang Anda rencanakan untuk dipanggil. Saat menjalankan fungsi Lambda`FunctionName`,,`InvocationType`, `LogType` dan merupakan parameter yang digunakan untuk memanggil metode `invoke` pada objek layanan Lambda.

Saat meneruskan parameter ke panggilan metode objek layanan, berikan objek JSON ke panggilan metode, seperti yang ditunjukkan pada contoh berikut untuk menjalankan fungsi 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);
   }
});
```

Mengembalikan Data sebagai JSON

JSON menyediakan cara standar untuk meneruskan data antara bagian-bagian dari aplikasi yang perlu mengirim beberapa nilai pada saat yang sama. Metode kelas klien di API biasanya mengembalikan JSON dalam `data` parameter yang diteruskan ke fungsi callback mereka. Misalnya, berikut adalah panggilan ke `getBucketCors` metode kelas klien 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));
  }
});
```

Nilai `data` adalah objek JSON, dalam contoh ini JSON yang menjelaskan konfigurasi CORS saat ini untuk bucket Amazon S3 yang ditentukan.

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

Coba lagi strategi di v2 AWS SDK untuk JavaScript

Banyak komponen di jaringan, seperti server DNS, sakelar, penyeimbang beban, dan lainnya dapat menghasilkan kesalahan di mana pun selama permintaan tertentu. Teknik biasa untuk menangani

respons kesalahan ini dalam lingkungan jaringan adalah dengan menerapkan percobaan ulang dalam aplikasi klien. Teknik ini meningkatkan keandalan aplikasi dan mengurangi biaya operasional bagi pengembang. AWS SDKs menerapkan logika coba ulang otomatis untuk AWS permintaan Anda.

Perilaku coba lagi berbasis backoff eksponensial

AWS SDK untuk JavaScript v2 mengimplementasikan logika coba lagi menggunakan backoff eksponensial dengan jitter penuh untuk kontrol aliran yang lebih baik. Ide di balik backoff eksponensial adalah menggunakan waktu tunggu yang semakin lama antara percobaan ulang untuk respons kesalahan yang berurutan. Jitter (penundaan acak) digunakan untuk mencegah tabrakan berturut-turut.

Menguji coba lagi penundaan di v2

Untuk menguji penundaan coba lagi di v2, kode di node_ modules/aws-sdk/lib/event _listeners.js diperbarui ke `console.log` nilai yang ada dalam penundaan variabel sebagai berikut:

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

Coba lagi penundaan dengan konfigurasi default

Anda dapat menguji penundaan untuk operasi apa pun pada klien AWS SDK. Kami memanggil `listTables` operasi pada klien DynamoDB menggunakan kode berikut:

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

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

Untuk menguji percobaan ulang, kami mensimulasikan `NetworkingError` dengan memutuskan koneksi internet dari perangkat yang menjalankan kode pengujian. Anda juga dapat mengatur proxy untuk mengembalikan Kesalahan kustom.

Saat menjalankan kode, Anda dapat melihat bahwa coba lagi penundaan menggunakan backoff eksponensial dengan jitter sebagai berikut:

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

Karena coba lagi menggunakan jitter, Anda akan mendapatkan nilai yang berbeda dalam menjalankan kode contoh.

Coba lagi penundaan dengan basis kustom

AWS SDK untuk JavaScript V2 memungkinkan melewatkan jumlah basis khusus milidetik untuk digunakan dalam backoff eksponensial untuk percobaan ulang operasi. Defaultnya 100 ms untuk semua layanan kecuali DynamoDB, di mana defaultnya 50 ms.

Kami menguji coba ulang dengan basis khusus 1000 ms sebagai berikut:

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

Kami mensimulasikan `NetworkingError` dengan memutuskan koneksi internet dari perangkat yang menjalankan kode pengujian. Anda dapat melihat bahwa nilai untuk penundaan coba lagi lebih tinggi dibandingkan dengan proses sebelumnya di mana defaultnya adalah 50 atau 100 ms.

```
retry delay: 356.2841549249l3
retry delay: 1183.5216495444615
retry delay: 2266.997988094194
retry delay: 1244.6948354966453
retry delay: 4200.323030066383
```

Karena coba lagi menggunakan jitter, Anda akan mendapatkan nilai yang berbeda dalam menjalankan kode contoh.

Coba lagi penundaan dengan algoritma backoff khusus

AWS SDK untuk JavaScript V2 juga memungkinkan meneruskan fungsi backoff khusus yang menerima hitungan coba lagi dan kesalahan dan mengembalikan jumlah waktu untuk menunda dalam milidetik. Jika hasilnya adalah nilai negatif bukan nol, tidak ada upaya coba lagi yang akan dilakukan.

Kami menguji fungsi backoff khusus yang menggunakan backoff linier dengan nilai dasar 200 ms sebagai berikut:

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

Kami mensimulasikan `NetworkingError` dengan memutuskan koneksi internet dari perangkat yang menjalankan kode pengujian. Anda dapat melihat bahwa nilai untuk penundaan coba lagi adalah kelipatan 200.

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

SDK untuk Contoh JavaScript Kode

Topik di bagian ini berisi contoh bagaimana menggunakan AWS SDK untuk JavaScript dengan APIs berbagai layanan untuk melaksanakan tugas-tugas umum.

Temukan kode sumber untuk contoh-contoh ini dan lainnya dalam repositori contoh kode AWS dokumentasi di. GitHub Untuk mengusulkan contoh kode baru agar tim AWS dokumentasi mempertimbangkan untuk memproduksi, buat permintaan baru. Tim ingin menghasilkan contoh kode yang mencakup skenario dan kasus penggunaan yang lebih luas, dibandingkan cuplikan kode sederhana yang hanya mencakup panggilan API individual. Untuk petunjuk, lihat bagian Penulisan kode di Pedoman kontribusi.

Topik

- CloudWatch Contoh Amazon
- Contoh Amazon DynamoDB
- EC2 Contoh Amazon
- AWS Elemental MediaConvert Contoh
- AWS Contoh IAM
- Contoh Kinesis Amazon
- Contoh-contoh Amazon S3
- Contoh Layanan Email Sederhana Amazon
- Contoh Layanan Pemberitahuan Sederhana Amazon
- Amazon SQS Contoh

CloudWatch Contoh Amazon

Amazon CloudWatch (CloudWatch) adalah layanan web yang memantau sumber daya Amazon Web Services dan aplikasi yang Anda jalankan AWS secara real time. Anda dapat menggunakan CloudWatch untuk mengumpulkan dan melacak metrik, yang merupakan variabel yang dapat Anda ukur untuk sumber daya dan aplikasi Anda. CloudWatch alarm mengirim pemberitahuan atau secara otomatis membuat perubahan pada sumber daya yang Anda pantau berdasarkan aturan yang Anda tetapkan.

JavaScript API untuk CloudWatch diekspos melalui kelas `AWS.CloudWatchAWS.CloudWatchEvents`,, dan `AWS.CloudWatchLogs` klien. Untuk informasi selengkapnya tentang penggunaan class CloudWatch klien `Class: AWS.CloudWatch`, lihat `Class: AWS.CloudWatchEvents`, dan `Class: AWS.CloudWatchLogs`di referensi API.

Topik

- Membuat Alarm di Amazon CloudWatch
- Menggunakan Alarm Actions di Amazon CloudWatch
- Mendapatkan Metrik dari Amazon CloudWatch
- Mengirim Acara ke CloudWatch Acara Amazon
- Menggunakan Filter Berlangganan di CloudWatch Log Amazon

Membuat Alarm di Amazon CloudWatch

Contoh kode Node.js ini menunjukkan:

- Cara mengambil informasi dasar tentang CloudWatch alarm Anda.
- Cara membuat dan menghapus CloudWatch alarm.

Skenario

Alarm mengawasi satu metrik selama jangka waktu yang Anda tentukan, dan melakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas tertentu selama jangka waktu tertentu.

Dalam contoh ini, serangkaian modul Node.js digunakan untuk membuat alarm di CloudWatch. Modul Node.js menggunakan SDK JavaScript untuk membuat alarm menggunakan metode kelas `AWS.CloudWatch` klien berikut:

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

Untuk informasi selengkapnya tentang CloudWatch alarm, lihat Membuat CloudWatch Alarm Amazon di CloudWatch Panduan Pengguna Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Menggambarkan Alarm

Buat modul Node.js dengan nama file`cw_describealarms.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch, buat objek `AWS.CloudWatch` layanan. Buat objek JSON untuk menahan parameter untuk mengambil deskripsi alarm, membatasi alarm yang dikembalikan ke alarm dengan status. `INSUFFICIENT_DATA` Kemudian panggil `describeAlarms` metode objek `AWS.CloudWatch` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cw_describealarms.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat Alarm untuk CloudWatch Metrik

Buat modul Node.js dengan nama file`cw_putmetricalarm.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch, buat objek `AWS.CloudWatch` layanan. Buat objek JSON untuk parameter yang diperlukan untuk membuat alarm berdasarkan metrik, dalam hal ini pemanfaatan CPU dari instance Amazon EC2 . Parameter yang tersisa diatur sehingga alarm terpicu ketika metrik melebihi ambang 70 persen. Kemudian panggil `describeAlarms` metode objek `AWS.CloudWatch` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cw_putmetricalarm.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Alarm

Buat modul Node.js dengan nama file`cw_deletealarms.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch, buat objek `AWS.CloudWatch` layanan. Buat objek JSON untuk menyimpan nama alarm yang ingin Anda hapus. Kemudian panggil `deleteAlarms` metode objek `AWS.CloudWatch` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cw_deletealarms.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Alarm Actions di Amazon CloudWatch

Contoh kode Node.js ini menunjukkan:

- Cara mengubah status EC2 instans Amazon Anda secara otomatis berdasarkan CloudWatch alarm.

Skenario

Dengan menggunakan tindakan alarm, Anda dapat membuat alarm yang secara otomatis menghentikan, menghentikan, me-reboot, atau memulihkan instans Amazon EC2 Anda. Anda dapat menggunakan tindakan berhenti atau menghentikan saat Anda tidak lagi membutuhkan instance untuk dijalankan. Anda dapat menggunakan tindakan reboot dan memulihkan untuk secara otomatis me-reboot instance tersebut.

Dalam contoh ini, serangkaian modul Node.js digunakan untuk menentukan tindakan alarm CloudWatch yang memicu reboot EC2 instance Amazon. Modul Node.js menggunakan SDK for JavaScript untuk mengelola EC2 instans Amazon menggunakan metode kelas `CloudWatch` klien berikut:

- `enableAlarmActions`
- `disableAlarmActions`

Untuk informasi selengkapnya tentang tindakan CloudWatch alarm, lihat Membuat Alarm untuk Berhenti, Menghentikan, Memulai Ulang, atau Memulihkan Instance di CloudWatch Panduan Pengguna Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat peran IAM yang kebijakannya memberikan izin untuk mendeskripsikan, mem-boot ulang, menghentikan, atau menghentikan instans Amazon. EC2 Untuk informasi selengkapnya tentang membuat peran IAM, lihat Membuat Peran untuk Mendelegasikan Izin ke AWS Layanan di Panduan Pengguna IAM.

Gunakan kebijakan peran berikut saat membuat peran IAM.

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

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

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global lalu setel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur keus-west-2.

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

Membuat dan Mengaktifkan Tindakan pada Alarm

Buat modul Node.js dengan nama filecw_enablealarmactions.js. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch, buat objek AWS.CloudWatch layanan.

Buat objek JSON untuk menahan parameter untuk membuat alarm, menentukan ActionsEnabled sebagai true dan array ARNs untuk tindakan yang akan dipicu alarm. Panggil putMetricAlarm metode objek AWS.CloudWatch layanan, yang membuat alarm jika tidak ada atau memperbaruinya jika alarm memang ada.

Dalam fungsi callback untukputMetricAlarm, setelah berhasil menyelesaikan membuat objek JSON yang berisi nama alarm. CloudWatch Panggil enableAlarmActions metode untuk mengaktifkan tindakan alarm.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cw_enablealarmactions.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menonaktifkan Tindakan pada Alarm

Buat modul Node.js dengan nama file`cw_disablealarmactions.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch, buat objek `AWS.CloudWatch` layanan. Buat objek JSON yang berisi nama CloudWatch alarm. Panggil `disableAlarmActions` metode untuk menonaktifkan tindakan untuk alarm ini.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cw_disablealarmactions.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mendapatkan Metrik dari Amazon CloudWatch

Contoh kode Node.js ini menunjukkan:

- Cara mengambil daftar CloudWatch metrik yang diterbitkan.

- Cara mempublikasikan titik data ke CloudWatch metrik.

Skenario

Metrik adalah data tentang performa sistem Anda. Anda dapat mengaktifkan pemantauan terperinci dari beberapa sumber daya, seperti EC2 instans Amazon Anda, atau metrik aplikasi Anda sendiri.

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mendapatkan metrik dari CloudWatch dan mengirim peristiwa ke Amazon CloudWatch Events. Modul Node.js menggunakan SDK JavaScript untuk mendapatkan metrik dari CloudWatch menggunakan metode kelas `CloudWatch` klien berikut:

- `listMetrics`
- `putMetricData`

Untuk informasi selengkapnya tentang CloudWatch metrik, lihat Menggunakan CloudWatch Metrik Amazon di CloudWatch Panduan Pengguna Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Metrik Daftar

Buat modul Node.js dengan nama file`cw_listmetrics.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch, buat objek `AWS.CloudWatch` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk membuat daftar metrik dalam namespace. `AWS/Logs` Panggil `listMetrics` metode untuk membuat daftar `IncomingLogEvents` metrik.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cw_listmetrics.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengirimkan Metrik Kustom

Buat modul Node.js dengan nama file`cw_putmetricdata.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch, buat objek `AWS.CloudWatch` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mengirimkan titik data untuk metrik `PAGES_VISITED` kustom. Panggil metode `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));
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cw_putmetricdata.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengirim Acara ke CloudWatch Acara Amazon

Contoh kode Node.js ini menunjukkan:

- Cara membuat dan memperbarui aturan yang digunakan untuk memicu peristiwa.
- Bagaimana mendefinisikan satu atau lebih target untuk menanggapi suatu peristiwa.
- Cara mengirim peristiwa yang dicocokkan dengan target untuk ditangani.

Skenario

CloudWatch Acara memberikan aliran peristiwa sistem yang mendekati real-time yang menjelaskan perubahan sumber daya Amazon Web Services ke salah satu dari berbagai target. Dengan menggunakan aturan sederhana, Anda dapat mencocokkan acara dan meruteknya ke satu atau beberapa fungsi atau aliran target.

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengirim peristiwa ke CloudWatch Acara. Modul Node.js menggunakan SDK JavaScript untuk mengelola instance menggunakan metode kelas `CloudWatchEvents` klien berikut:

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

Untuk informasi selengkapnya tentang CloudWatch Acara, lihat Menambahkan Acara dengan PutEvents di Panduan Pengguna CloudWatch Acara Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat fungsi Lambda menggunakan cetak biru hello-world untuk dijadikan target acara. Untuk mempelajari caranya, lihat Langkah 1: Membuat AWS Lambda fungsi di Panduan Pengguna CloudWatch Acara Amazon.
- Buat peran IAM yang kebijakannya memberikan izin untuk CloudWatch Acara dan yang termasuk `events.amazonaws.com` sebagai entitas tepercaya. Untuk informasi selengkapnya tentang

membuat peran IAM, lihat Membuat Peran untuk Mendelegasikan Izin ke AWS Layanan di Panduan Pengguna IAM.

Gunakan kebijakan peran berikut saat membuat peran IAM.

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

Gunakan hubungan kepercayaan berikut saat membuat peran IAM.

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

Membuat Aturan Terjadwal

Buat modul Node.js dengan nama file`cwe_putrule.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch Acara, buat objek `AWS.CloudWatchEvents` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menentukan aturan terjadwal baru, yang meliputi:

- Nama untuk aturan
- ARN dari peran IAM yang Anda buat sebelumnya
- Ekspresi untuk menjadwalkan pemicu aturan setiap lima menit

Panggil `putRule` metode untuk membuat aturan. Callback mengembalikan ARN dari aturan baru atau yang diperbarui.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cwe_putrule.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menambahkan Target AWS Lambda Fungsi

Buat modul Node.js dengan nama filecwe_puttargets.js. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch Acara, buat objek `AWS.CloudWatchEvents` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menentukan aturan yang ingin Anda lampirkan target, termasuk ARN dari fungsi Lambda yang Anda buat. Panggil `putTargets` metode objek `AWS.CloudWatchEvents` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cwe_puttargets.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengirim Acara

Buat modul Node.js dengan nama file`cwe_putevents.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch Acara, buat objek `AWS.CloudWatchEvents` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mengirim acara. Untuk setiap acara, sertakan sumber acara, sumber daya apa pun yang terpengaruh oleh acara tersebut, dan detail untuk acara tersebut. ARNs Panggil `putEvents` metode objek `AWS.CloudWatchEvents` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cwe_putevents.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Filter Berlangganan di CloudWatch Log Amazon

Contoh kode Node.js ini menunjukkan:

- Cara membuat dan menghapus filter untuk peristiwa log di CloudWatch Log.

Skenario

Langganan menyediakan akses ke umpan real-time peristiwa CloudWatch log dari Log dan mengirimkan umpan tersebut ke layanan lain, seperti aliran Amazon Kinesis AWS Lambda atau, untuk pemrosesan, analisis, atau pemuatan kustom ke sistem lain. Filter langganan menentukan pola yang akan digunakan untuk memfilter peristiwa log mana yang dikirimkan ke sumber daya Anda AWS .

Dalam contoh ini, serangkaian modul Node.js digunakan untuk membuat daftar, membuat, dan menghapus filter langganan di CloudWatch Log. Tujuan untuk peristiwa log adalah fungsi Lambda. Modul Node.js menggunakan SDK JavaScript untuk mengelola filter langganan menggunakan metode kelas `CloudWatchLogs` klien berikut:

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

Untuk informasi selengkapnya tentang langganan CloudWatch Log, lihat Pemrosesan Data Log secara real-time dengan Langganan di Panduan Pengguna CloudWatch Log Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat fungsi Lambda sebagai tujuan untuk peristiwa log. Anda harus menggunakan ARN dari fungsi ini. Untuk informasi selengkapnya tentang menyiapkan fungsi Lambda, lihat Filter Langganan dengan AWS Lambda di Panduan Pengguna CloudWatch Log Amazon.
- Buat peran IAM yang kebijakannya memberikan izin untuk menjalankan fungsi Lambda yang Anda buat dan memberikan akses penuh ke CloudWatch Log atau menerapkan kebijakan berikut ke peran eksekusi yang Anda buat untuk fungsi Lambda. Untuk informasi selengkapnya tentang membuat peran IAM, lihat Membuat Peran untuk Mendelegasikan Izin ke AWS Layanan di Panduan Pengguna IAM.

Gunakan kebijakan peran berikut saat membuat peran IAM.

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

Menjelaskan Filter Langganan yang Ada

Buat modul Node.js dengan nama file`cwl_describesubscriptionfilters.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch Log, buat objek `AWS.CloudWatchLogs` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mendeskripsikan filter yang ada, termasuk nama grup log dan jumlah maksimum filter yang ingin Anda jelaskan. Panggil metode `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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cwl_describesubscriptionfilters.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat Filter Langganan

Buat modul Node.js dengan nama file`cwl_putsubscriptionfilter.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch Log, buat objek `AWS.CloudWatchLogs` layanan. Buat objek JSON yang berisi parameter yang diperlukan

untuk membuat filter, termasuk ARN dari fungsi Lambda tujuan, nama filter, pola string untuk pemfilteran, dan nama grup log. Panggil metode `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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cwl_putsubscriptionfilter.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Filter Langganan

Buat modul Node.js dengan nama file`cwl_deletesubscriptionfilters.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses CloudWatch Log, buat objek `AWS.CloudWatchLogs` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus filter, termasuk nama filter dan grup log. Panggil metode `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);
  }
});
```

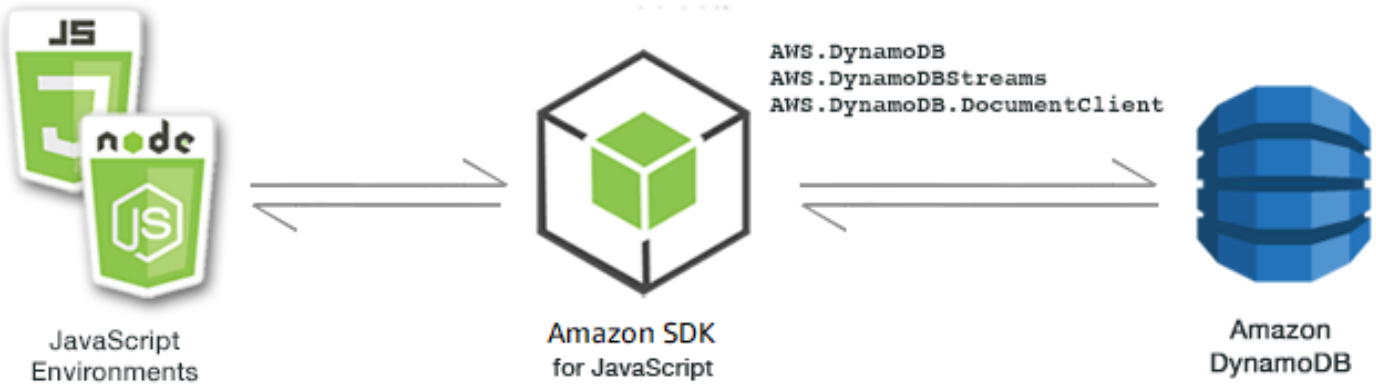
Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node cwl_deletesubscriptionfilter.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Contoh Amazon DynamoDB

Amazon DynamoDB adalah database cloud NoSQL yang dikelola sepenuhnya yang mendukung model penyimpanan dokumen dan nilai kunci. Anda membuat tabel tanpa skema untuk data tanpa perlu menyediakan atau memelihara server database khusus.



JavaScript API untuk DynamoDB diekspos melalui kelas`AWS.DynamoDBStreams`,, `AWS.DynamoDB` `AWS.DynamoDB.DocumentClient` dan klien. Untuk informasi selengkapnya tentang penggunaan kelas klien DynamoDB, `Class: AWS.DynamoDB`lihat `Class: AWS.DynamoDBStreams`, `Class: AWS.DynamoDB.DocumentClient`dan di referensi API.

Topik

- Membuat dan Menggunakan Tabel di DynamoDB
- Membaca dan Menulis Satu Item di DynamoDB
- Membaca dan Menulis Item dalam Batch di DynamoDB
- Meminta dan Memindai Tabel DynamoDB
- Menggunakan Klien Dokumen DynamoDB

Membuat dan Menggunakan Tabel di DynamoDB

Contoh kode Node.js ini menunjukkan:

- Cara membuat dan mengelola tabel yang digunakan untuk menyimpan dan mengambil data dari DynamoDB.

Skenario

Mirip dengan sistem basis data lainnya, DynamoDB menyimpan data dalam tabel. Tabel DynamoDB adalah kumpulan data yang disusun ke dalam item yang analog dengan baris. Untuk menyimpan atau mengakses data di DynamoDB, Anda membuat dan bekerja dengan tabel.

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk melakukan operasi dasar dengan tabel DynamoDB. Kode menggunakan SDK for JavaScript untuk membuat dan bekerja dengan tabel dengan menggunakan metode kelas `AWS.DynamoDB` klien berikut:

- CreateTable
- ListTables
- DescribeTable

- Deletable

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Membuat Tabel

Buat modul Node.js dengan nama file`ddb_createtable.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk membuat tabel, yang dalam contoh ini mencakup nama dan tipe data untuk setiap atribut, skema kunci, nama tabel, dan unit throughput untuk penyediaan. Panggil `createTable` metode objek layanan 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_createtable.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Tabel Anda

Buat modul Node.js dengan nama file`ddb_listtables.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk daftar tabel Anda, yang dalam contoh ini membatasi jumlah tabel yang terdaftar ke 10. Panggil `listTables` metode objek layanan 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_listtables.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menjelaskan Tabel

Buat modul Node.js dengan nama file`ddb_describetable.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menggambarkan tabel, yang dalam contoh ini mencakup nama tabel yang disediakan sebagai parameter baris perintah. Panggil `describeTable` metode objek layanan 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_describetable.js TABLE_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Tabel

Buat modul Node.js dengan nama file`ddb_deletetable.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus tabel, yang dalam contoh ini mencakup nama tabel yang disediakan sebagai parameter baris perintah. Panggil `deleteTable` metode objek layanan 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_deletetable.js TABLE_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membaca dan Menulis Satu Item di DynamoDB

Contoh kode Node.js ini menunjukkan:

- Cara menambahkan item dalam tabel DynamoDB.
- Cara mengambil item dalam tabel DynamoDB.
- Cara menghapus item dalam tabel DynamoDB.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk membaca dan menulis satu item dalam tabel DynamoDB dengan menggunakan metode kelas klien ini: `AWS.DynamoDB`

- putItem
- GetItem
- DeleteItem

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat tabel DynamoDB yang itemnya dapat Anda akses. Untuk informasi selengkapnya tentang membuat tabel DynamoDB, lihat. Membuat dan Menggunakan Tabel di DynamoDB

Menulis Item

Buat modul Node.js dengan nama file`ddb_putitem.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menambahkan item, yang dalam contoh ini mencakup nama tabel dan peta yang mendefinisikan atribut yang akan ditetapkan dan nilai untuk setiap atribut. Panggil `putItem` metode objek layanan 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_putitem.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mendapatkan Item

Buat modul Node.js dengan nama fileddb_getitem.js. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat AWS.DynamoDB objek layanan. Untuk mengidentifikasi item yang akan didapat, Anda harus memberikan nilai kunci utama untuk item tersebut dalam tabel. Secara default, getItem metode mengembalikan semua nilai atribut didefinisikan untuk item. Untuk mendapatkan hanya subset dari semua nilai atribut yang mungkin, tentukan ekspresi proyeksi.

Buat objek JSON yang berisi parameter yang diperlukan untuk mendapatkan item, yang dalam contoh ini mencakup nama tabel, nama dan nilai kunci untuk item yang Anda dapatkan, dan ekspresi proyeksi yang mengidentifikasi atribut item yang ingin Anda ambil. Panggil getItem metode objek layanan 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);
  }
```

```
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_getitem.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Item

Buat modul Node.js dengan nama file`ddb_deleteitem.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus item, yang dalam contoh ini mencakup nama tabel dan nama kunci dan nilai untuk item yang Anda hapus. Panggil `deleteItem` metode objek layanan 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_deleteitem.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membaca dan Menulis Item dalam Batch di DynamoDB

Contoh kode Node.js ini menunjukkan:

- Cara membaca dan menulis batch item dalam tabel DynamoDB.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk menempatkan sekumpulan item dalam tabel DynamoDB serta membaca sekumpulan item. Kode menggunakan SDK for JavaScript untuk melakukan operasi baca dan tulis batch menggunakan metode kelas klien DynamoDB berikut:

- batchGetItem
- batchWriteItem

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat tabel DynamoDB yang itemnya dapat Anda akses. Untuk informasi selengkapnya tentang membuat tabel DynamoDB, lihat. Membuat dan Menggunakan Tabel di DynamoDB

Membaca Item dalam Batch

Buat modul Node.js dengan nama file`ddb_batchgetitem.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mendapatkan sekumpulan item, yang dalam contoh ini mencakup nama satu atau lebih tabel dari mana untuk membaca, nilai-nilai kunci untuk dibaca di setiap tabel, dan ekspresi proyeksi yang menentukan atribut untuk kembali. Panggil `batchGetItem` metode objek layanan 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);
    });
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_batchgetitem.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menulis Item dalam Batch

Buat modul Node.js dengan nama fileddb_batchwriteitem.js. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat AWS.DynamoDB objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mendapatkan sekumpulan item, yang dalam contoh ini mencakup tabel tempat Anda ingin menulis item, kunci yang ingin Anda tulis untuk setiap item, dan atribut beserta nilainya. Panggil batchWriteItem metode objek layanan 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_batchwriteitem.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Meminta dan Memindai Tabel DynamoDB

Contoh kode Node.js ini menunjukkan:

- Cara menanyakan dan memindai tabel DynamoDB untuk item.

Skenario

Query menemukan item dalam tabel atau indeks sekunder hanya menggunakan nilai atribut kunci primer. Anda harus memberikan nama kunci partisi dan nilai yang harus dicari. Anda juga dapat memberikan nama dan nilai kunci pengurutan, dan menggunakan operator perbandingan untuk menyempurnakan hasil pencarian. Pemindaian menemukan item dengan memeriksa setiap item dalam tabel yang ditentukan.

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk mengidentifikasi satu atau beberapa item yang ingin Anda ambil dari tabel DynamoDB. Kode menggunakan SDK for JavaScript untuk melakukan kueri dan memindai tabel menggunakan metode kelas klien DynamoDB berikut:

- query
- memindai

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat tabel DynamoDB yang itemnya dapat Anda akses. Untuk informasi selengkapnya tentang membuat tabel DynamoDB, lihat. Membuat dan Menggunakan Tabel di DynamoDB

Melakukan Kueri Tabel

Contoh ini menanyakan tabel yang berisi informasi episode tentang seri video, mengembalikan judul episode dan subtitle episode musim kedua melewati episode 9 yang berisi frasa tertentu dalam subtitle mereka.

Buat modul Node.js dengan nama file`ddb_query.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menanyakan tabel, yang dalam contoh ini mencakup nama tabel, yang `ExpressionAttributeValues` dibutuhkan oleh kueri, a `KeyConditionExpression` yang menggunakan nilai-nilai tersebut untuk menentukan item mana yang dikembalikan kueri, dan nama nilai atribut yang akan dikembalikan untuk setiap item. Panggil `query` metode objek layanan 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 + ")");
    });
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_query.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memindai Tabel

Buat modul Node.js dengan nama file`ddb_scan.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat `AWS.DynamoDB` objek layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk memindai tabel untuk item, yang dalam contoh ini mencakup nama tabel, daftar nilai atribut yang akan dikembalikan untuk setiap item yang cocok, dan ekspresi untuk memfilter set hasil untuk menemukan item yang berisi frasa tertentu. Panggil `scan` metode objek layanan 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 + ")"
      );
    });
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddb_scan.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Klien Dokumen DynamoDB

Contoh kode Node.js ini menunjukkan:

- Cara mengakses tabel DynamoDB menggunakan klien dokumen.

Skenario

Klien dokumen DynamoDB menyederhanakan bekerja dengan item dengan mengabstraksi gagasan nilai atribut. Abstraksi ini menganotasi JavaScript tipe asli yang disediakan sebagai parameter input, serta mengonversi data respons beranotasi ke tipe asli. JavaScript

Untuk informasi selengkapnya tentang class DynamoDB Document Client, `AWS.DynamoDB.DocumentClient`lihat di Referensi API. Untuk informasi selengkapnya tentang pemrograman dengan Amazon DynamoDB, lihat Pemrograman dengan DynamoDB di Panduan Pengembang Amazon DynamoDB.

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk melakukan operasi dasar pada tabel DynamoDB menggunakan klien dokumen. Kode menggunakan SDK for JavaScript untuk query dan scan tabel menggunakan metode kelas DynamoDB Document Client ini:

- mendapatkan
- menempatkan
- perbarui
- query
- menghapus

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat tabel DynamoDB yang itemnya dapat Anda akses. Untuk informasi selengkapnya tentang membuat tabel DynamoDB menggunakan SDK JavaScript for, lihat. Membuat dan Menggunakan Tabel di DynamoDB Anda juga dapat menggunakan konsol DynamoDB untuk membuat tabel.

Mendapatkan Item dari Tabel

Buat modul Node.js dengan nama file`ddbdoc_get.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat objek. `AWS.DynamoDB.DocumentClient` Buat objek JSON yang berisi parameter yang dibutuhkan dapatkan item dari tabel, yang dalam contoh ini mencakup nama tabel, nama kunci hash dalam tabel itu, dan nilai kunci hash untuk item yang ingin Anda dapatkan. Panggil `get` metode klien dokumen 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddbdoc_get.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menempatkan Item di Tabel

Buat modul Node.js dengan nama file`ddbdoc_put.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat objek. `AWS.DynamoDB.DocumentClient` Buat objek JSON yang berisi parameter yang diperlukan untuk menulis item ke tabel, yang dalam contoh ini mencakup nama tabel dan deskripsi item untuk ditambahkan atau diperbarui yang mencakup hashkey dan nilai serta nama dan nilai untuk atribut untuk diatur pada item. Panggil `put` metode klien dokumen 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddbdoc_put.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memperbarui Item dalam Tabel

Buat modul Node.js dengan nama fileddbdoc_update.js. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat objek. AWS.DynamoDB.DocumentClient Buat objek JSON yang berisi parameter yang diperlukan untuk menulis item ke tabel, yang dalam contoh ini mencakup nama tabel, kunci item yang akan diperbarui, satu set UpdateExpressions yang menentukan atribut item yang akan diperbarui dengan token yang Anda tetapkan nilai dalam parameter. ExpressionAttributeValues Panggil update metode klien dokumen 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);
```

```
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddbdoc_update.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Melakukan Kueri Tabel

Contoh ini menanyakan tabel yang berisi informasi episode tentang seri video, mengembalikan judul episode dan subtitle episode musim kedua melewati episode 9 yang berisi frasa tertentu dalam subtitle mereka.

Buat modul Node.js dengan nama file`ddbdoc_query.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat objek. `AWS.DynamoDB.DocumentClient` Buat objek JSON yang berisi parameter yang diperlukan untuk menanyakan tabel, yang dalam contoh ini mencakup nama tabel, yang `ExpressionAttributeValues` dibutuhkan oleh kueri, dan `KeyConditionExpression` yang menggunakan nilai-nilai tersebut untuk menentukan item mana yang dikembalikan kueri. Panggil `query` metode klien dokumen 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddbdoc_query.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Item dari Tabel

Buat modul Node.js dengan nama file`ddbdoc_delete.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses DynamoDB, buat objek. `AWS.DynamoDB.DocumentClient` Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus item dalam tabel, yang dalam contoh ini mencakup nama tabel serta nama dan nilai hashkey item yang ingin Anda hapus. Panggil `delete` metode klien dokumen 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ddbdoc_delete.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

EC2 Contoh Amazon

Amazon Elastic Compute Cloud (Amazon EC2) adalah layanan web yang menyediakan hosting server virtual di cloud. Ini dirancang untuk membuat komputasi awan skala web lebih mudah bagi pengembang dengan menyediakan kapasitas komputasi yang dapat diubah ukurannya.

JavaScript API untuk Amazon EC2 diekspos melalui kelas `AWS.EC2` klien. Untuk informasi selengkapnya tentang menggunakan kelas EC2 klien Amazon, lihat `Class: AWS.EC2`di referensi API.

Topik

- Membuat EC2 Instans Amazon
- Mengelola EC2 Instans Amazon
- Bekerja dengan Amazon EC2 Key Pairs
- Menggunakan Wilayah dan Availability Zone dengan Amazon EC2
- Bekerja dengan Grup Keamanan di Amazon EC2
- Menggunakan Alamat IP Elastis di Amazon EC2

Membuat EC2 Instans Amazon

Contoh kode Node.js ini menunjukkan:

- Cara membuat EC2 instance Amazon dari Amazon Machine Image (AMI) publik.
- Cara membuat dan menetapkan tag ke EC2 instance Amazon baru.

Tentang Contoh

Dalam contoh ini, Anda menggunakan modul Node.js untuk membuat EC2 instance Amazon dan menetapkan key pair dan tag ke dalamnya. Kode menggunakan SDK for JavaScript untuk membuat dan menandai instance dengan menggunakan metode kelas EC2 klien Amazon berikut:

- `runInstances`
- `createTags`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini.

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Membuat key pair. Untuk detailnya, lihat Bekerja dengan Amazon EC2 Key Pairs. Anda menggunakan nama key pair dalam contoh ini.

Membuat dan Menandai Instance

Buat modul Node.js dengan nama file`ec2_createinstances.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan parameter untuk `runInstances` metode AWS. EC2 class client, termasuk nama key pair yang akan ditetapkan dan ID AMI yang akan dijalankan. Untuk memanggil `runInstances` metode ini, buat janji untuk memanggil objek EC2 layanan Amazon, melewati parameter. Kemudian tangani respons dalam panggilan balik janji.

Kode selanjutnya menambahkan `Name` tag ke instance baru, yang dikenali dan ditampilkan oleh EC2 konsol Amazon di bidang Nama daftar instance. Anda dapat menambahkan hingga 50 tag ke sebuah instance, yang semuanya dapat ditambahkan dalam satu panggilan ke `createTags` metode.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_createinstances.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengelola EC2 Instans Amazon

Contoh kode Node.js ini menunjukkan:

- Cara mengambil informasi dasar tentang EC2 instans Amazon Anda.
- Cara memulai dan menghentikan pemantauan terperinci dari EC2 instans Amazon.
- Cara memulai dan menghentikan EC2 instance Amazon.
- Cara me-reboot EC2 instance Amazon.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk melakukan beberapa operasi manajemen instance dasar. Modul Node.js menggunakan SDK for JavaScript untuk mengelola instance dengan menggunakan metode kelas EC2 klien Amazon ini:

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

Untuk informasi selengkapnya tentang siklus hidup EC2 instans Amazon, lihat Siklus Hidup Instance di Panduan Pengguna Amazon. EC2

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat EC2 instance Amazon. Untuk informasi selengkapnya tentang membuat EC2 instans Amazon, lihat EC2 Instans Amazon di Panduan Pengguna Amazon EC2 atau EC2 Instans Amazon di Panduan Pengguna Amazon EC2 .

Menjelaskan Instance Anda

Buat modul Node.js dengan nama file`ec2_describeinstances.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Panggil `describeInstances` metode objek EC2 layanan Amazon untuk mengambil deskripsi rinci tentang instance Anda.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_describeinstances.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengelola Pemantauan Instance

Buat modul Node.js dengan nama file`ec2_monitorinstances.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Tambahkan instance instance IDs yang ingin Anda kontrol pemantauan.

Berdasarkan nilai argumen baris perintah (`ON`atau`OFF`), panggil salah satu `monitorInstances` metode objek EC2 layanan Amazon untuk memulai pemantauan terperinci dari instance yang ditentukan atau memanggil metode. un`monitorInstances` Gunakan `DryRun` parameter untuk menguji apakah Anda memiliki izin untuk mengubah pemantauan instans sebelum Anda mencoba mengubah pemantauan instance ini.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah, tentukan `ON` untuk memulai pemantauan terperinci atau `OFF` untuk menghentikan pemantauan.

```
node ec2_monitorinstances.js ON
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memulai dan Menghentikan Instans

Buat modul Node.js dengan nama file`ec2_startstopinstances.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Tambahkan instance IDs dari instance yang ingin Anda mulai atau hentikan.

Berdasarkan nilai argumen baris perintah (`START`atau`STOP`), panggil `startInstances` metode objek EC2 layanan Amazon untuk memulai instance yang ditentukan, atau `stopInstances` metode untuk menghentikannya. Gunakan `DryRun` parameter untuk menguji apakah Anda memiliki izin sebelum benar-benar mencoba memulai atau menghentikan instance yang dipilih.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah yang menentukan `START` untuk memulai instance atau `STOP` menghentikannya.

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

Kode contoh ini dapat ditemukan di sini GitHub.

Mem-boot Ulang Instans

Buat modul Node.js dengan nama file`ec2_rebootinstances.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek EC2 layanan Amazon. Tambahkan instance instance IDs yang ingin Anda reboot. Panggil `rebootInstances` metode objek `AWS.EC2` layanan untuk me-reboot instance yang ditentukan. Gunakan `DryRun` parameter untuk menguji apakah Anda memiliki izin untuk me-reboot instance ini sebelum benar-benar mencoba untuk me-reboot mereka.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_rebootinstances.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Bekerja dengan Amazon EC2 Key Pairs

Contoh kode Node.js ini menunjukkan:

- Cara mengambil informasi tentang pasangan kunci Anda.

- Cara membuat key pair untuk mengakses EC2 instance Amazon.
- Cara menghapus key pair yang ada.

Skenario

Amazon EC2 menggunakan kriptografi kunci publik untuk mengenkripsi dan mendekripsi informasi login. Kriptografi kunci publik menggunakan kunci publik untuk mengenkripsi data, kemudian penerima menggunakan kunci pribadi untuk mendekripsi data. Kunci publik dan privat dikenal sebagai pasangan kunci.

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk melakukan beberapa operasi manajemen EC2 key pair Amazon. Modul Node.js menggunakan SDK for JavaScript untuk mengelola instance dengan menggunakan metode kelas EC2 klien Amazon berikut:

- `createKeyPair`
- `deleteKeyPair`
- `describeKeyPairs`

Untuk informasi selengkapnya tentang pasangan EC2 kunci Amazon, lihat Pasangan EC2 Kunci Amazon di Panduan EC2 Pengguna Amazon atau Pasangan EC2 Kunci Amazon dan Instans Windows di Panduan EC2 Pengguna Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Menggambarkan Pasangan Kunci Anda

Buat modul Node.js dengan nama file`ec2_describekeypairs.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat objek JSON kosong untuk menahan parameter yang dibutuhkan oleh `describeKeyPairs` metode untuk mengembalikan deskripsi untuk semua pasangan kunci Anda.

Anda juga dapat memberikan array nama pasangan kunci di `KeyName` bagian parameter dalam file JSON ke `describeKeyPairs` metode.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_describekeypairs.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat Pasangan Kunci

Setiap key pair membutuhkan nama. Amazon EC2 mengaitkan kunci publik dengan nama yang Anda tentukan sebagai nama kunci. Buat modul Node.js dengan nama file`ec2_createkeypair.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat parameter JSON untuk menentukan nama key pair, lalu berikan mereka untuk memanggil `createKeyPair` metode.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_createkeypair.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Pasangan Kunci

Buat modul Node.js dengan nama file`ec2_deletekeypair.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat parameter JSON untuk menentukan nama key pair yang ingin Anda hapus. Kemudian panggil `deleteKeyPair` metodenya.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_deletekeypair.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Wilayah dan Availability Zone dengan Amazon EC2

Contoh kode Node.js ini menunjukkan:

- Cara mengambil deskripsi untuk Wilayah dan Zona Ketersediaan.

Skenario

Amazon EC2 di-host di beberapa lokasi di seluruh dunia. Lokasi ini terdiri dari Wilayah dan Zona Ketersediaan . Setiap Wilayah adalah wilayah geografis yang terpisah. Setiap Wilayah memiliki beberapa lokasi terisolasi yang dikenal sebagai Zona Ketersediaan. Amazon EC2 menyediakan kemampuan untuk menempatkan instance dan data di beberapa lokasi.

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk mengambil detail tentang Regions dan Availability Zones. Modul Node.js menggunakan SDK for JavaScript untuk mengelola instance dengan menggunakan metode berikut dari kelas EC2 klien Amazon:

- `describeAvailabilityZones`
- `describeRegions`

Untuk informasi selengkapnya tentang Wilayah dan Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan di Panduan EC2 Pengguna Amazon atau Wilayah dan Zona Ketersediaan di Panduan EC2 Pengguna Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Menjelaskan Wilayah dan Availability Zone

Buat modul Node.js dengan nama file`ec2_describeregionsandzones.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat objek JSON kosong untuk diteruskan sebagai parameter, yang mengembalikan semua deskripsi yang tersedia. Kemudian panggil `describeRegions` dan `describeAvailabilityZones` metode.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_describeregionsandzones.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Bekerja dengan Grup Keamanan di Amazon EC2

Contoh kode Node.js ini menunjukkan:

- Cara mengambil informasi tentang grup keamanan Anda.
- Cara membuat grup keamanan untuk mengakses EC2 instans Amazon.
- Cara menghapus grup keamanan yang ada.

Skenario

Grup EC2 keamanan Amazon bertindak sebagai firewall virtual yang mengontrol lalu lintas untuk satu atau lebih contoh. Anda menambahkan aturan ke setiap grup keamanan untuk mengizinkan lalu lintas ke atau dari instans terkait. Anda dapat mengubah aturan untuk grup keamanan kapan saja; aturan baru diterapkan secara otomatis ke semua instance yang terkait dengan grup keamanan.

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk melakukan beberapa EC2 operasi Amazon yang melibatkan grup keamanan. Modul Node.js menggunakan SDK for JavaScript untuk mengelola instance dengan menggunakan metode berikut dari kelas EC2 klien Amazon:

- `describeSecurityGroups`
- `authorizeSecurityGroupIngress`
- `createSecurityGroup`

- `describeVpcs`
- `deleteSecurityGroup`

Untuk informasi selengkapnya tentang grup EC2 keamanan Amazon, lihat Grup Keamanan EC2 Amazon Amazon untuk Instans Linux di Panduan EC2 Pengguna Amazon atau Grup EC2 Keamanan Amazon untuk Instans Windows di Panduan EC2 Pengguna Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Menggambarkan Grup Keamanan Anda

Buat modul Node.js dengan nama file`ec2_describesecuritygroups.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat objek JSON untuk diteruskan sebagai parameter, termasuk grup IDs untuk grup keamanan yang ingin Anda jelaskan. Kemudian panggil `describeSecurityGroups` metode objek EC2 layanan 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));
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_describesecuritygroups.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat Grup Keamanan dan Aturan

Buat modul Node.js dengan nama file`ec2_createsecuritygroup.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat objek JSON untuk parameter yang menentukan nama grup keamanan, deskripsi, dan ID untuk VPC. Lewati parameter ke `createSecurityGroup` metode.

Setelah berhasil membuat grup keamanan, Anda dapat menentukan aturan untuk mengizinkan lalu lintas masuk. Buat objek JSON untuk parameter yang menentukan protokol IP dan port masuk tempat EC2 instance Amazon akan menerima lalu lintas. Lewati parameter ke `authorizeSecurityGroupIngress` metode.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_createsecuritygroup.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Grup Keamanan

Buat modul Node.js dengan nama file`ec2_deletesecuritygroup.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat parameter JSON untuk menentukan nama grup keamanan yang akan dihapus. Kemudian panggil `deleteSecurityGroup` metodenya.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_deletesecuritygroup.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Alamat IP Elastis di Amazon EC2

Contoh kode Node.js ini menunjukkan:

- Cara mengambil deskripsi alamat IP Elastis Anda.
- Cara mengalokasikan dan melepaskan alamat IP elastis.
- Cara mengaitkan alamat IP Elastis dengan EC2 instans Amazon.

Skenario

Alamat IP Elastis adalah alamat IP statis yang dirancang untuk komputasi awan dinamis. Alamat IP Elastis dikaitkan dengan AWS akun Anda. Ini adalah alamat IP publik, yang dapat dijangkau dari Internet. Jika instans Anda tidak memiliki alamat IP publik, Anda dapat mengaitkan alamat IP Elastis dengan instans Anda untuk mengaktifkan komunikasi dengan Internet.

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk melakukan beberapa EC2 operasi Amazon yang melibatkan alamat IP Elastis. Modul Node.js menggunakan SDK for JavaScript untuk mengelola alamat IP Elastic dengan menggunakan metode kelas EC2 klien Amazon berikut:

- `describeAddresses`
- `allocateAddress`
- `associateAddress`
- `releaseAddress`

Untuk informasi selengkapnya tentang alamat IP Elastis di Amazon EC2, lihat Alamat IP Elastis di Panduan EC2 Pengguna Amazon atau Alamat IP Elastis di Panduan EC2 Pengguna Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat EC2 instance Amazon. Untuk informasi selengkapnya tentang membuat EC2 instans Amazon, lihat EC2 Instans Amazon di Panduan EC2 Pengguna Amazon atau EC2 Instans Amazon di Panduan Pengguna Amazon EC2 .

Menjelaskan Alamat IP Elastis

Buat modul Node.js dengan nama file`ec2_describeaddresses.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat objek JSON untuk diteruskan sebagai parameter, memfilter alamat yang dikembalikan oleh yang ada di VPC Anda. Untuk mengambil deskripsi semua alamat IP Elastis Anda, hilangkan filter dari parameter JSON. Kemudian panggil `describeAddresses` metode objek EC2 layanan 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));
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_describeaddresses.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengalokasikan dan Mengaitkan Alamat IP Elastis dengan Instans Amazon EC2

Buat modul Node.js dengan nama file`ec2_allocateaddress.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat objek JSON untuk parameter yang digunakan untuk mengalokasikan alamat IP Elastis,

yang dalam hal ini menentukan `Domain` adalah VPC. Panggil `allocateAddress` metode objek EC2 layanan Amazon.

Jika panggilan berhasil, `data` parameter ke fungsi callback memiliki `AllocationId` properti yang mengidentifikasi alamat IP Elastis yang dialokasikan.

Buat objek JSON untuk parameter yang digunakan untuk mengaitkan alamat IP Elastis ke EC2 instans Amazon, termasuk `AllocationId` dari alamat yang baru dialokasikan dan instance Amazon EC2 . `InstanceId` Kemudian panggil `associateAddresses` metode objek EC2 layanan 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);
      }
    });
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_allocateaddress.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Melepaskan Alamat IP Elastis

Buat modul Node.js dengan nama file`ec2_releaseaddress.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon EC2, buat objek `AWS.EC2` layanan. Buat objek JSON untuk parameter yang digunakan untuk melepaskan alamat IP Elastis, yang dalam hal ini menentukan `AllocationId` untuk alamat IP Elastis. Melepaskan alamat IP Elastis juga memisahkannya dari instans Amazon apa pun. EC2 Panggil `releaseAddress` metode objek EC2 layanan 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");
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_releaseaddress.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

AWS Elemental MediaConvert Contoh

AWS Elemental MediaConvert adalah layanan transcoding video berbasis file dengan fitur tingkat siaran. Anda dapat menggunakannya untuk membuat aset untuk siaran dan untuk pengiriman video-on-demand (VOD) di internet. Untuk informasi selengkapnya, lihat Panduan Pengguna AWS Elemental MediaConvert .

JavaScript API untuk MediaConvert diekspos melalui kelas `AWS.MediaConvert` klien. Untuk informasi selengkapnya, lihat `Class: AWS.MediaConvert`di referensi API.

Topik

- Mendapatkan Titik Akhir Khusus Wilayah Anda untuk MediaConvert
- Membuat dan Mengelola Pekerjaan Transcoding di MediaConvert
- Menggunakan Job Template di MediaConvert

Mendapatkan Titik Akhir Khusus Wilayah Anda untuk MediaConvert

Contoh kode Node.js ini menunjukkan:

- Cara mengambil titik akhir khusus wilayah Anda dari. MediaConvert

Skenario

Dalam contoh ini, Anda menggunakan modul Node.js untuk memanggil MediaConvert dan mengambil titik akhir khusus wilayah Anda. Anda dapat mengambil URL titik akhir Anda dari titik akhir default layanan sehingga belum memerlukan titik akhir khusus wilayah Anda. Kode menggunakan SDK for JavaScript untuk mengambil titik akhir ini dengan menggunakan metode kelas klien ini: MediaConvert

- `describeEndpoints`

> **Important**
>
> Agen HTTP/HTTPS Node.js default membuat koneksi TCP baru untuk setiap permintaan baru. Untuk menghindari biaya membangun koneksi baru, AWS SDK untuk JavaScript menggunakan kembali koneksi TCP. Untuk informasi selengkapnya, lihat Menggunakan Kembali Koneksi dengan Keep-Alive di Node.js.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat peran IAM yang memberikan MediaConvert akses ke file input Anda dan bucket Amazon S3 tempat file output Anda disimpan. Untuk detailnya, lihat Mengatur Izin IAM di AWS Elemental MediaConvert Panduan Pengguna.

Mendapatkan URL Endpoint Anda

Buat modul Node.js dengan nama file`emc_getendpoint.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk melewati parameter permintaan kosong untuk `describeEndpoints` metode kelas `AWS.MediaConvert` klien. Untuk memanggil `describeEndpoints` metode, buat janji untuk memanggil objek MediaConvert layanan, melewati parameter. Tangani respons dalam callback janji.

```
// Load the SDK for JavaScript.
const aws = require("aws-sdk");

// Set the AWS Region.
aws.config.update({ region: "us-west-2" });

// Create the client.
const mediaConvert = new aws.MediaConvert({ apiVersion: "2017-08-29" });
```

```
exports.handler = async (event, context) => {
  // Create empty request parameters
  const params = {
    MaxResults: 0,
  };

  try {
    const { Endpoints } = await mediaConvert
      .describeEndpoints(params)
      .promise();
    console.log("Your MediaConvert endpoint is ", Endpoints);
  } catch (err) {
    console.log("MediaConvert Error", err);
  }
};
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node emc_getendpoint.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat dan Mengelola Pekerjaan Transcoding di MediaConvert

Contoh kode Node.js ini menunjukkan:

- Cara menentukan titik akhir khusus wilayah yang akan digunakan. MediaConvert
- Cara membuat pekerjaan transcoding di MediaConvert.
- Cara membatalkan pekerjaan transcoding.
- Cara mengambil JSON untuk pekerjaan transcoding yang selesai.
- Cara mengambil array JSON hingga 20 pekerjaan yang paling baru dibuat.

Skenario

Dalam contoh ini, Anda menggunakan modul Node.js untuk memanggil MediaConvert untuk membuat dan mengelola pekerjaan transcoding. Kode menggunakan SDK JavaScript untuk melakukan ini dengan menggunakan metode kelas MediaConvert klien berikut:

- `createJob`
- `cancelJob`
- `getJob`
- `listJobs`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat dan konfigurasikan bucket Amazon S3 yang menyediakan penyimpanan untuk file input pekerjaan dan file output. Untuk detailnya, lihat Membuat Penyimpanan untuk File di Panduan AWS Elemental MediaConvert Pengguna.
- Unggah video input ke bucket Amazon S3 yang Anda sediakan untuk penyimpanan input. Untuk daftar codec dan kontainer video masukan yang didukung, lihat Codec dan Container Input yang Didukung di Panduan Pengguna.AWS Elemental MediaConvert
- Buat peran IAM yang memberikan MediaConvert akses ke file input Anda dan bucket Amazon S3 tempat file output Anda disimpan. Untuk detailnya, lihat Mengatur Izin IAM di AWS Elemental MediaConvert Panduan Pengguna.

Mengkonfigurasi SDK

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global, lalu setel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur ke`us-west-2`. Karena MediaConvert menggunakan titik akhir khusus untuk setiap akun, Anda juga harus mengonfigurasi kelas `AWS.MediaConvert` klien untuk menggunakan titik akhir khusus wilayah Anda. Untuk melakukan ini, atur `endpoint` parameter pada`AWS.config.mediaconvert`.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };
```

Mendefinisikan Job Transcoding Sederhana

Buat modul Node.js dengan nama file`mc_createjob.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat JSON yang mendefinisikan parameter pekerjaan transcode.

Parameter ini cukup rinci. Anda dapat menggunakan AWS Elemental MediaConvert konsol untuk menghasilkan parameter pekerjaan JSON dengan memilih pengaturan pekerjaan di konsol, lalu memilih Tampilkan pekerjaan JSON di bagian bawah bagian Job. Contoh ini menunjukkan JSON untuk pekerjaan sederhana.

```
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",
    },
  },
};
```

Membuat Job Transcoding

Setelah membuat parameter pekerjaan JSON, panggil `createJob` metode dengan membuat janji untuk memanggil objek `AWS.MediaConvert` layanan, melewati parameter. Kemudian tangani respons dalam panggilan balik janji. ID pekerjaan yang dibuat dikembalikan dalam responsdata.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node emc_createjob.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membatalkan Job Transcoding

Buat modul Node.js dengan nama fileemc_canceljob.js. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat JSON yang menyertakan ID pekerjaan yang akan dibatalkan. Kemudian panggil cancelJob metode dengan membuat janji untuk memanggil objek AWS.MediaConvert layanan, melewati parameter. Tangani respons dalam callback janji.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ec2_canceljob.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Lowongan Transcoding Terbaru

Buat modul Node.js dengan nama file`mc_listjobs.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat parameter JSON, termasuk nilai untuk menentukan apakah akan mengurutkan daftar dalam`ASCENDING`, atau `DESCENDING` urutan, ARN antrian pekerjaan yang akan diperiksa, dan status pekerjaan yang akan disertakan. Kemudian panggil `listJobs` metode dengan membuat janji untuk memanggil objek `AWS.MediaConvert` layanan, melewati parameter. Tangani respons dalam callback janji.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node emc_listjobs.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Job Template di MediaConvert

Contoh kode Node.js ini menunjukkan:

- Cara membuat template MediaConvert pekerjaan.
- Cara menggunakan template pekerjaan untuk membuat pekerjaan transcoding.
- Cara membuat daftar semua templat pekerjaan Anda.
- Cara menghapus template pekerjaan.

Skenario

JSON yang diperlukan untuk membuat pekerjaan transcoding MediaConvert secara rinci, berisi sejumlah besar pengaturan. Anda dapat sangat menyederhanakan penciptaan lapangan kerja dengan menyimpan pengaturan yang diketahui baik dalam templat pekerjaan yang dapat Anda gunakan untuk membuat pekerjaan berikutnya. Dalam contoh ini, Anda menggunakan modul Node.js untuk memanggil MediaConvert untuk membuat, menggunakan, dan mengelola template pekerjaan. Kode menggunakan SDK JavaScript untuk melakukan ini dengan menggunakan metode kelas MediaConvert klien berikut:

- `createJobTemplate`
- `createJob`
- `deleteJobTemplate`
- `listJobTemplates`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat peran IAM yang memberikan MediaConvert akses ke file input Anda dan bucket Amazon S3 tempat file output Anda disimpan. Untuk detailnya, lihat Mengatur Izin IAM di AWS Elemental MediaConvert Panduan Pengguna.

Membuat Template Job

Buat modul Node.js dengan nama file`emc_create_jobtemplate.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Tentukan parameter JSON untuk pembuatan template. Anda dapat menggunakan sebagian besar parameter JSON dari pekerjaan sebelumnya yang berhasil untuk menentukan `Settings` nilai dalam template. Contoh ini menggunakan pengaturan pekerjaan dariMembuat dan Mengelola Pekerjaan Transcoding di MediaConvert.

Panggil `createJobTemplate` metode dengan membuat janji untuk memanggil objek `AWS.MediaConvert` layanan, melewati parameter. Kemudian tangani respons dalam panggilan balik janji.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node emc_create_jobtemplate.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat Transcoding Job dari Job Template

Buat modul Node.js dengan nama file`emc_template_createjob.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat parameter pembuatan pekerjaan JSON, termasuk nama template pekerjaan yang akan digunakan, dan penggunaan yang khusus untuk pekerjaan yang Anda buat. `Settings` Kemudian panggil `createJobs` metode dengan membuat janji untuk memanggil objek `AWS.MediaConvert` layanan, melewati parameter. Tangani respons dalam callback janji.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node emc_template_createjob.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Template Job Anda

Buat modul Node.js dengan nama file`emc_listtemplates.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk melewati parameter permintaan untuk `listTemplates` metode kelas `AWS.MediaConvert` klien. Sertakan nilai untuk menentukan templat apa yang akan dicantumkan (`NAME`,`CREATION DATE`,`SYSTEM`), berapa banyak yang akan dicantumkan, dan urutan urutannya. Untuk memanggil `listTemplates` metode, buat janji untuk memanggil objek MediaConvert layanan, melewati parameter. Kemudian tangani respons dalam panggilan balik janji.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node emc_listtemplates.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Template Job

Buat modul Node.js dengan nama file`emc_deletetemplate.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan nama template pekerjaan yang ingin Anda hapus sebagai parameter untuk `deleteJobTemplate` metode kelas `AWS.MediaConvert` klien. Untuk memanggil

`deleteJobTemplate` metode, buat janji untuk memanggil objek MediaConvert layanan, melewati parameter. Tangani respons dalam callback janji.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node emc_deletetemplate.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

AWS Contoh IAM

AWS Identity and Access Management (IAM) adalah layanan web yang memungkinkan pelanggan Amazon Web Services untuk mengelola pengguna dan izin pengguna di. AWS Layanan ini ditargetkan pada organisasi dengan banyak pengguna atau sistem di cloud yang menggunakan AWS produk. Dengan IAM, Anda dapat mengelola pengguna secara terpusat, kredensyal keamanan

seperti kunci akses, dan izin yang mengontrol sumber daya mana AWS yang dapat diakses pengguna.

JavaScript API untuk IAM diekspos melalui kelas `AWS.IAM` klien. Untuk informasi selengkapnya tentang penggunaan kelas klien IAM, lihat `Class: AWS.IAM`di referensi API.

Topik

- Mengelola Pengguna IAM
- Bekerja dengan Kebijakan IAM
- Mengelola Kunci Akses IAM
- Bekerja dengan Sertifikat Server IAM
- Mengelola Alias Akun IAM

Mengelola Pengguna IAM

Contoh kode Node.js ini menunjukkan:

- Cara mengambil daftar pengguna IAM.
- Cara membuat dan menghapus pengguna.
- Cara memperbarui nama pengguna.

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk membuat dan mengelola pengguna di IAM. Modul Node.js menggunakan SDK JavaScript untuk membuat, menghapus, dan memperbarui pengguna menggunakan metode kelas `AWS.IAM` klien berikut:

- `createUser`
- `listUsers`
- `updateUser`
- `getUser`
- `deleteUser`

Untuk informasi selengkapnya tentang pengguna IAM, lihat Pengguna IAM di Panduan Pengguna IAM.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Membuat Pengguna

Buat modul Node.js dengan nama file`iam_createuser.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan, yang terdiri dari nama pengguna yang ingin Anda gunakan untuk pengguna baru sebagai parameter baris perintah.

Panggil `getUser` metode objek `AWS.IAM` layanan untuk melihat apakah nama pengguna sudah ada. Jika nama pengguna saat ini tidak ada, panggil `createUser` metode untuk membuatnya. Jika nama sudah ada, tulis pesan untuk efek itu ke konsol.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_createuser.js USER_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Pengguna di Akun Anda

Buat modul Node.js dengan nama file`iam_listusers.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mencantumkan pengguna Anda, membatasi jumlah yang dikembalikan dengan menyetel `MaxItems` parameter ke 10. Panggil `listUsers` metode objek `AWS.IAM` layanan. Tulis nama pengguna pertama dan tanggal pembuatan ke konsol.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_listusers.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memperbarui Nama Pengguna

Buat modul Node.js dengan nama file`iam_updateuser.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mencantumkan pengguna Anda, dengan menentukan nama pengguna saat ini dan baru sebagai parameter baris perintah. Panggil `updateUser` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah, tentukan nama pengguna saat ini diikuti dengan nama pengguna baru.

```
node iam_updateuser.js ORIGINAL_USERNAME NEW_USERNAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Pengguna

Buat modul Node.js dengan nama file`iam_deleteuser.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan, yang terdiri dari nama pengguna yang ingin Anda hapus sebagai parameter baris perintah.

Panggil `getUser` metode objek `AWS.IAM` layanan untuk melihat apakah nama pengguna sudah ada. Jika nama pengguna saat ini tidak ada, tulis pesan untuk efek itu ke konsol. Jika pengguna ada, panggil `deleteUser` metode untuk menghapusnya.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_deleteuser.js USER_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Bekerja dengan Kebijakan IAM

Contoh kode Node.js ini menunjukkan:

- Cara membuat dan menghapus kebijakan IAM.
- Cara melampirkan dan melepaskan kebijakan IAM dari peran.

Skenario

Anda memberikan izin kepada pengguna dengan membuat kebijakan, yaitu dokumen yang mencantumkan tindakan yang dapat dilakukan pengguna dan sumber daya yang dapat memengaruhi

tindakan tersebut. Setiap tindakan atau sumber daya yang tidak diizinkan secara eksplisit ditolak secara default. Kebijakan dapat dibuat dan dilampirkan ke pengguna, grup pengguna, peran yang diambil oleh pengguna, dan sumber daya.

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengelola kebijakan di IAM. Modul Node.js menggunakan SDK JavaScript untuk membuat dan menghapus kebijakan serta melampirkan dan melepaskan kebijakan peran menggunakan metode kelas klien berikut`AWS.IAM`:

- `createPolicy`
- `getPolicy`
- `listAttachedRolePolicies`
- `attachRolePolicy`
- `detachRolePolicy`

Untuk informasi selengkapnya tentang pengguna IAM, lihat Ringkasan Manajemen Akses: Izin dan Kebijakan di Panduan Pengguna IAM.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Buat peran IAM yang dapat Anda lampirkan kebijakan. Untuk informasi selengkapnya tentang membuat peran, lihat Membuat Peran IAM di Panduan Pengguna IAM.

Membuat Kebijakan IAM

Buat modul Node.js dengan nama file`iam_createpolicy.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat dua objek JSON, satu berisi dokumen kebijakan yang ingin Anda buat dan yang lainnya berisi parameter yang diperlukan untuk membuat kebijakan, yang mencakup JSON kebijakan dan nama yang ingin Anda berikan kebijakan. Pastikan untuk melakukan stringifikasi objek JSON kebijakan dalam parameter. Panggil `createPolicy` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_createpolicy.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mendapatkan Kebijakan IAM

Buat modul Node.js dengan nama file`iam_getpolicy.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mengambil kebijakan, yang merupakan ARN dari kebijakan yang ingin Anda dapatkan. Panggil `getPolicy` metode objek `AWS.IAM` layanan. Tulis deskripsi kebijakan ke konsol.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_getpolicy.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Melampirkan Kebijakan Peran Terkelola

Buat modul Node.js dengan nama file`iam_attachrolepolicy.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mendapatkan daftar kebijakan IAM terkelola yang dilampirkan ke peran, yang terdiri dari nama peran. Berikan nama peran sebagai parameter baris perintah. Panggil `listAttachedRolePolicies` metode objek `AWS.IAM` layanan, yang mengembalikan array kebijakan terkelola ke fungsi callback.

Periksa anggota array untuk melihat apakah kebijakan yang ingin Anda lampirkan ke peran sudah dilampirkan. Jika kebijakan tidak dilampirkan, panggil `attachRolePolicy` metode untuk melampirkannya.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_attachrolepolicy.js IAM_ROLE_NAME
```

Melepaskan Kebijakan Peran Terkelola

Buat modul Node.js dengan nama file`iam_detachrolepolicy.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mendapatkan daftar kebijakan IAM terkelola yang dilampirkan ke peran, yang terdiri dari nama peran. Berikan nama peran sebagai parameter baris perintah. Panggil `listAttachedRolePolicies` metode objek `AWS.IAM` layanan, yang mengembalikan array kebijakan terkelola dalam fungsi callback.

Periksa anggota array untuk melihat apakah kebijakan yang ingin Anda lepaskan dari peran terlampir. Jika kebijakan dilampirkan, panggil `detachRolePolicy` metode untuk melepaskannya.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_detachrolepolicy.js IAM_ROLE_NAME
```

Mengelola Kunci Akses IAM

Contoh kode Node.js ini menunjukkan:

- Cara mengelola kunci akses pengguna Anda.

Skenario

Pengguna memerlukan kunci akses mereka sendiri untuk melakukan panggilan terprogram AWS dari SDK untuk. JavaScript Untuk memenuhi kebutuhan ini, Anda dapat membuat, memodifikasi, melihat, atau memutar kunci akses (kunci akses IDs dan kunci akses rahasia) untuk pengguna IAM. Secara

default, saat Anda membuat kunci akses, statusnya adalah`Active`, yang berarti pengguna dapat menggunakan kunci akses untuk panggilan API.

Dalam contoh ini, serangkaian modul Node.js digunakan mengelola kunci akses di IAM. Modul Node.js menggunakan SDK JavaScript untuk mengelola kunci akses IAM menggunakan metode kelas `AWS.IAM` klien berikut:

- createAccessKey
- listAccessKeys
- getAccessKeyLastUsed
- updateAccessKey
- deleteAccessKey

Untuk informasi selengkapnya tentang kunci akses IAM, lihat Kunci Akses di Panduan Pengguna IAM.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Membuat Kunci Akses untuk Pengguna

Buat modul Node.js dengan nama file`iam_createaccesskeys.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk membuat kunci akses baru, yang mencakup nama pengguna IAM. Panggil `createAccessKey` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Pastikan untuk menyalurkan data yang dikembalikan ke file teks agar tidak kehilangan kunci rahasia, yang hanya dapat diberikan sekali.

```
node iam_createaccesskeys.js > newuserkeys.txt
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Kunci Akses Pengguna

Buat modul Node.js dengan nama file`iam_listaccesskeys.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mengambil kunci akses pengguna, yang mencakup nama pengguna IAM dan opsional jumlah maksimum pasangan kunci akses yang ingin Anda cantumkan. Panggil `listAccessKeys` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_listaccesskeys.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mendapatkan Penggunaan Terakhir untuk Kunci Akses

Buat modul Node.js dengan nama file`iam_accesskeylastused.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk membuat kunci akses baru, yang merupakan ID kunci akses yang Anda inginkan informasi penggunaan terakhir. Panggil `getAccessKeyLastUsed` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_accesskeylastused.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memperbarui Status Kunci Akses

Buat modul Node.js dengan nama file`iam_updateaccesskey.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk memperbarui status kunci akses, yang mencakup ID kunci akses dan status yang diperbarui. Statusnya bisa `Active` atau`Inactive`. Panggil `updateAccessKey` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_updateaccesskey.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Kunci Akses

Buat modul Node.js dengan nama file`iam_deleteaccesskey.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus kunci akses, yang mencakup ID kunci akses dan nama pengguna. Panggil `deleteAccessKey` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_deleteaccesskey.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Bekerja dengan Sertifikat Server IAM

Contoh kode Node.js ini menunjukkan:

- Bagaimana melakukan tugas-tugas dasar dalam mengelola sertifikat server untuk koneksi HTTPS.

Skenario

Untuk mengaktifkan koneksi HTTPS ke situs web atau aplikasi Anda AWS, Anda memerlukan sertifikat server SSL/TLS. Untuk menggunakan sertifikat yang Anda peroleh dari penyedia eksternal dengan situs web atau aplikasi Anda AWS, Anda harus mengunggah sertifikat ke IAM atau mengimpornya ke AWS Certificate Manager.

Dalam contoh ini, serangkaian modul Node.js digunakan untuk menangani sertifikat server di IAM. Modul Node.js menggunakan SDK JavaScript untuk mengelola sertifikat server menggunakan metode kelas `AWS.IAM` klien berikut:

- listServerCertificates
- getServerCertificate
- updateServerCertificate
- deleteServerCertificate

Untuk informasi selengkapnya tentang sertifikat server, lihat Bekerja dengan Sertifikat Server di Panduan Pengguna IAM.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensyal bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Cantumkan Sertifikat Server Anda

Buat modul Node.js dengan nama file`iam_listservercerts.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Panggil `listServerCertificates` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_listservercerts.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mendapatkan Sertifikat Server

Buat modul Node.js dengan nama file`iam_getservercert.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan dapatkan sertifikat, yang terdiri dari nama sertifikat server yang Anda inginkan. Panggil `getServerCertificates` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_getservercert.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memperbarui Sertifikat Server

Buat modul Node.js dengan nama file`iam_updateservercert.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk memperbarui sertifikat, yang terdiri dari nama sertifikat server yang ada serta nama sertifikat baru. Panggil `updateServerCertificate` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_updateservercert.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Sertifikat Server

Buat modul Node.js dengan nama file`iam_deleteservercert.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus sertifikat server, yang terdiri dari nama sertifikat yang ingin Anda hapus. Panggil `deleteServerCertificates` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_deleteservercert.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengelola Alias Akun IAM

Contoh kode Node.js ini menunjukkan:

- Cara mengelola alias untuk ID AWS akun Anda.

Skenario

Jika Anda ingin URL untuk halaman masuk berisi nama perusahaan atau pengenal ramah lainnya, bukan ID AWS akun Anda, Anda dapat membuat alias untuk ID akun Anda AWS . Jika Anda membuat alias AWS akun, URL halaman masuk Anda berubah untuk memasukkan alias.

Dalam contoh ini, serangkaian modul Node.js digunakan untuk membuat dan mengelola alias akun IAM. Modul Node.js menggunakan SDK JavaScript untuk mengelola alias menggunakan metode kelas `AWS.IAM` klien berikut:

- createAccountAlias
- listAccountAliases
- deleteAccountAlias

Untuk informasi selengkapnya tentang alias akun IAM, lihat ID AWS Akun Anda dan Aliasnya di Panduan Pengguna IAM.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensyal bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Membuat Akun Alias

Buat modul Node.js dengan nama file`iam_createaccountalias.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk membuat alias akun, yang mencakup alias yang ingin Anda buat. Panggil `createAccountAlias` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_createaccountalias.js ALIAS
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Alias Akun

Buat modul Node.js dengan nama file`iam_listaccountaliases.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek

`AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk daftar alias akun, yang mencakup jumlah maksimum item yang akan dikembalikan. Panggil `listAccountAliases` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_listaccountaliases.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Alias Akun

Buat modul Node.js dengan nama file`iam_deleteaccountalias.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses IAM, buat objek `AWS.IAM` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus alias akun, yang mencakup alias yang ingin Anda hapus. Panggil `deleteAccountAlias` metode objek `AWS.IAM` layanan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node iam_deleteaccountalias.js ALIAS
```

Kode contoh ini dapat ditemukan di sini GitHub.

Contoh Kinesis Amazon

Amazon Kinesis adalah platform untuk streaming data AWS, menawarkan layanan canggih untuk memuat dan menganalisis data streaming, dan juga menyediakan kemampuan bagi Anda untuk membangun aplikasi data streaming khusus untuk kebutuhan khusus.

JavaScript API untuk Kinesis diekspos melalui kelas `AWS.Kinesis` klien. Untuk informasi selengkapnya tentang penggunaan kelas klien Kinesis, lihat `Class: AWS.Kinesis`di referensi API.

Topik

- Menangkap Kemajuan Gulir Halaman Web dengan Amazon Kinesis

Menangkap Kemajuan Gulir Halaman Web dengan Amazon Kinesis

Contoh skrip browser ini menunjukkan:

- Cara menangkap progres gulir di halaman web dengan Amazon Kinesis sebagai contoh metrik penggunaan halaman streaming untuk analisis selanjutnya.

Skenario

Dalam contoh ini, halaman HTML sederhana mensimulasikan konten halaman blog. Saat pembaca menggulir posting blog yang disimulasikan, skrip browser menggunakan SDK untuk merekam jarak gulir JavaScript ke bawah halaman dan mengirim data tersebut ke Kinesis menggunakan metode `putRecords`kelas klien Kinesis. Data streaming yang diambil oleh Amazon Kinesis Data Streams kemudian dapat diproses oleh instans EC2 Amazon dan disimpan di salah satu dari beberapa penyimpanan data termasuk Amazon DynamoDB dan Amazon Redshift.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Buat aliran Kinesis. Anda perlu menyertakan ARN sumber daya aliran dalam skrip browser. Untuk informasi selengkapnya tentang membuat Amazon Kinesis Data Streams, lihat Mengelola Aliran Kinesis di Panduan Pengembang Amazon Kinesis Data Streams.
- Buat kumpulan identitas Amazon Cognito dengan akses diaktifkan untuk identitas yang tidak diautentikasi. Anda perlu menyertakan ID kumpulan identitas dalam kode untuk mendapatkan kredensyal untuk skrip browser. Untuk informasi selengkapnya tentang kumpulan identitas Amazon Cognito, lihat Kumpulan Identitas di Panduan Pengembang Amazon Cognito.
- Buat peran IAM yang kebijakannya memberikan izin untuk mengirimkan data ke aliran Kinesis. Untuk informasi selengkapnya tentang membuat peran IAM, lihat Membuat Peran untuk Mendelegasikan Izin ke AWS Layanan di Panduan Pengguna IAM.

Gunakan kebijakan peran berikut saat membuat peran IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "mobileanalytics:PutEvents",
            "cognito-sync:*"
         ],
         "Resource": [
            "*"
         ]
      },
      {
         "Effect": "Allow",
         "Action": [
            "kinesis:Put*"
         ],
         "Resource": [
            "STREAM_RESOURCE_ARN"
         ]
      }
   ]
}
```

Halaman Blog

HTML untuk halaman blog terutama terdiri dari serangkaian paragraf yang terkandung dalam suatu `<div>` elemen. Ketinggian yang dapat digulir `<div>` ini digunakan untuk membantu menghitung seberapa jauh pembaca telah menggulir konten saat mereka membaca. HTML juga berisi sepasang `<script>` elemen. Salah satu elemen ini menambahkan SDK untuk JavaScript ke halaman dan yang lainnya menambahkan skrip browser yang menangkap kemajuan gulir pada halaman dan melaporkannya ke 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>
```

Mengkonfigurasi SDK

Dapatkan kredensyal yang diperlukan untuk mengonfigurasi SDK dengan memanggil `CognitoIdentityCredentials` metode, dengan menyediakan ID kumpulan identitas Amazon Cognito. Setelah berhasil, buat objek layanan Kinesis dalam fungsi callback.

Cuplikan kode berikut menunjukkan langkah ini. (Lihat Menangkap Kode Kemajuan Gulir Halaman Web contoh lengkapnya.)

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

Membuat Catatan Gulir

Progres gulir dihitung menggunakan `scrollHeight` dan `scrollTop` properti yang `<div>` berisi konten posting blog. Setiap catatan gulir dibuat dalam fungsi pendengar acara untuk `scroll` acara tersebut dan kemudian ditambahkan ke array catatan untuk pengiriman berkala ke Kinesis.

Cuplikan kode berikut menunjukkan langkah ini. (Lihat Menangkap Kode Kemajuan Gulir Halaman Web contoh lengkapnya.)

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

Mengirimkan Catatan ke Kinesis

Sekali setiap detik, jika ada catatan dalam array, catatan yang tertunda dikirim ke Kinesis.

Cuplikan kode berikut menunjukkan langkah ini. (Lihat Menangkap Kode Kemajuan Gulir Halaman Web contoh lengkapnya.)

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

Menangkap Kode Kemajuan Gulir Halaman Web

Berikut adalah kode skrip browser untuk contoh kemajuan gulir halaman web Kinesis yang menangkap.

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

Contoh-contoh Amazon S3

Amazon Simple Storage Service (Amazon S3) adalah layanan web yang menyediakan penyimpanan cloud yang sangat skalabel. Amazon S3 menyediakan penyimpanan objek yang mudah digunakan, dengan antarmuka layanan web sederhana untuk menyimpan dan mengambil sejumlah data dari mana saja di web.

JavaScript API untuk Amazon S3 diekspos melalui kelas `AWS.S3` klien. Untuk informasi selengkapnya tentang menggunakan kelas klien Amazon S3, lihat `Class: AWS.S3`di referensi API.

Topik

- Contoh Browser Amazon S3
- Amazon S3 Node.js Contoh

Contoh Browser Amazon S3

Topik berikut menunjukkan dua contoh bagaimana AWS SDK untuk JavaScript dapat digunakan di browser untuk berinteraksi dengan bucket Amazon S3.

- Yang pertama menunjukkan skenario sederhana di mana foto yang ada di bucket Amazon S3 dapat dilihat oleh pengguna mana pun (tidak diautentikasi).
- Yang kedua menunjukkan skenario yang lebih kompleks di mana pengguna diizinkan untuk melakukan operasi pada foto di ember seperti mengunggah, menghapus, dll.

Topik

- Melihat Foto di Bucket Amazon S3 dari Browser
- Mengunggah Foto ke Amazon S3 dari Browser

Melihat Foto di Bucket Amazon S3 dari Browser

Contoh kode skrip browser ini menunjukkan:

- Cara membuat album foto di bucket Amazon Simple Storage Service (Amazon S3) Simple Storage S3) dan memungkinkan pengguna yang tidak diautentikasi untuk melihat foto.

Skenario

Dalam contoh ini, halaman HTML sederhana menyediakan aplikasi berbasis browser untuk melihat foto di album foto. Album foto ada dalam ember Amazon S3 tempat foto diunggah.

Skrip browser menggunakan SDK JavaScript untuk berinteraksi dengan bucket Amazon S3. Skrip menggunakan `listObjects`metode kelas klien Amazon S3 untuk memungkinkan Anda melihat album foto.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, pertama-tama selesaikan tugas-tugas ini.

> **Note**
>
> Dalam contoh ini, Anda harus menggunakan AWS Wilayah yang sama untuk bucket Amazon S3 dan kumpulan identitas Amazon Cognito.

Buat Bucket

Di konsol Amazon S3, buat bucket Amazon S3 tempat Anda dapat menyimpan album dan foto. Untuk informasi selengkapnya tentang menggunakan konsol untuk membuat bucket S3, lihat Membuat Bucket di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Saat Anda membuat bucket S3, pastikan untuk melakukan hal berikut:

- Catat nama bucket sehingga Anda dapat menggunakannya dalam tugas prasyarat berikutnya, Mengonfigurasi Izin Peran.
- Pilih AWS Region untuk membuat bucket. Ini harus Wilayah yang sama yang akan Anda gunakan untuk membuat kumpulan identitas Amazon Cognito dalam tugas prasyarat berikutnya, Buat Kumpulan Identitas.
- Konfigurasikan izin bucket dengan mengikuti Pengaturan izin untuk akses situs web di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Buat Identity Pool

Di konsol Amazon Cognito, buat kumpulan identitas Amazon Cognito, seperti yang dijelaskan the section called “Langkah 1: Buat Kolam Identitas Amazon Cognito” dalam topik Memulai dalam Skrip Browser.

Saat Anda membuat kumpulan identitas, catat nama kumpulan identitas, serta nama peran untuk identitas yang tidak diautentikasi.

Konfigurasikan Izin Peran

Untuk memungkinkan melihat album dan foto, Anda harus menambahkan izin ke peran IAM dari kumpulan identitas yang baru saja Anda buat. Mulailah dengan membuat kebijakan sebagai berikut.

1. Buka konsol IAM.
2. Di panel navigasi di sebelah kiri, pilih Kebijakan, lalu pilih tombol Buat kebijakan.
3. Pada tab JSON, masukkan definisi JSON berikut, tetapi ganti BUCKET_NAME dengan nama bucket.

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

```
      {
         "Effect": "Allow",
         "Action": [
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME"
         ]
      }
   ]
}
```

4. Pilih tombol Tinjau kebijakan, beri nama kebijakan dan berikan deskripsi (jika Anda mau), lalu pilih tombol Buat kebijakan.

   Pastikan untuk mencatat nama sehingga Anda dapat menemukannya dan melampirkannya ke peran IAM nanti.

Setelah kebijakan dibuat, navigasikan kembali ke konsol IAM. Temukan peran IAM untuk identitas tidak terautentikasi yang dibuat Amazon Cognito di tugas prasyarat sebelumnya, Buat Kumpulan Identitas. Anda menggunakan kebijakan yang baru saja dibuat untuk menambahkan izin ke identitas ini.

Meskipun alur kerja untuk tugas ini umumnya sama the section called “Langkah 2: Tambahkan Kebijakan ke Peran IAM yang Dibuat” dengan topik Memulai dalam Skrip Browser, ada beberapa perbedaan yang perlu diperhatikan:

- Gunakan kebijakan baru yang baru saja Anda buat, bukan kebijakan untuk Amazon Polly.
- Pada halaman Lampirkan Izin, untuk menemukan kebijakan baru dengan cepat, buka daftar Filter kebijakan dan pilih Pelanggan yang dikelola.

Untuk informasi tambahan tentang membuat peran IAM, lihat Membuat Peran untuk Mendelegasikan Izin ke AWS Layanan di Panduan Pengguna IAM.

Konfigurasikan CORS

Sebelum skrip browser dapat mengakses bucket Amazon S3, Anda harus mengatur konfigurasi CORS-nya sebagai berikut.

> **Important**
>
> Pada konsol S3 baru, konfigurasi CORS harus 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>
```

Buat Album dan Unggah Foto

Karena contoh ini hanya memungkinkan pengguna untuk melihat foto yang sudah ada di ember, Anda perlu membuat beberapa album di ember dan mengunggah foto ke mereka.

> **Note**
>
> Untuk contoh ini, nama file file foto harus dimulai dengan satu garis bawah (“_”). Karakter ini penting nanti untuk penyaringan. Selain itu, pastikan untuk menghormati hak cipta pemilik foto.

1. Di konsol Amazon S3, buka bucket yang Anda buat sebelumnya.
2. Pada tab Ikhtisar, pilih tombol Buat folder untuk membuat folder. Untuk contoh ini, beri nama folder “album1", “album2", dan “album3".
3. Untuk album1 dan kemudian album2, pilih folder dan kemudian unggah foto ke dalamnya sebagai berikut:
   a. Pilih tombol Unggah.
   b. Seret atau pilih file foto yang ingin Anda gunakan, lalu pilih Berikutnya.
   c. Di bawah Kelola izin publik, pilih Berikan akses baca publik ke objek ini.
   d. Pilih tombol Upload (di pojok kiri bawah).
4. Biarkan album3 kosong.

Mendefinisikan Halaman Web

HTML untuk aplikasi melihat foto terdiri dari `<div>` elemen di mana skrip browser membuat antarmuka tampilan. `<script>`Elemen pertama menambahkan SDK ke skrip browser. `<script>`Elemen kedua menambahkan JavaScript file eksternal yang memegang kode skrip browser.

Untuk contoh ini, file tersebut diberi nama`PhotoViewer.js`, dan terletak di folder yang sama dengan file HTML. Untuk menemukan SDK_VERSION_NUMBER saat ini, lihat Referensi API untuk SDK untuk Panduan Referensi API. JavaScript AWS SDK untuk 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>
```

Mengkonfigurasi SDK

Dapatkan kredensional yang Anda perlukan untuk mengonfigurasi SDK dengan memanggil metode. `CognitoIdentityCredentials` Anda harus memberikan ID kumpulan identitas Amazon Cognito. Kemudian buat objek `AWS.S3` layanan.

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

Sisa kode dalam contoh ini mendefinisikan fungsi-fungsi berikut untuk mengumpulkan dan menyajikan informasi tentang album dan foto dalam ember.

- `listAlbums`
- `viewAlbum`

Daftar Album di Bucket

Untuk mencantumkan semua album yang ada di bucket, `listAlbums` fungsi aplikasi memanggil `listObjects` metode objek `AWS.S3` layanan. Fungsi ini menggunakan `CommonPrefixes` properti sehingga panggilan hanya mengembalikan objek yang digunakan sebagai album (yaitu, folder).

Sisa fungsi mengambil daftar album dari bucket Amazon S3 dan menghasilkan HTML yang diperlukan untuk menampilkan daftar album di halaman 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);
```

```
    }
  });
}
```

Melihat Album

Untuk menampilkan konten album di bucket Amazon S3, `viewAlbum` fungsi aplikasi mengambil nama album dan membuat kunci Amazon S3 untuk album itu. Fungsi kemudian memanggil `listObjects` metode objek `AWS.S3` layanan untuk mendapatkan daftar semua objek (foto) dalam album.

Sisa fungsi mengambil daftar objek yang ada di album dan menghasilkan HTML yang diperlukan untuk menampilkan foto di halaman 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;");
  });
}
```

Melihat Foto di Bucket Amazon S3: Kode Lengkap

Bagian ini berisi HTML lengkap dan JavaScript kode untuk contoh di mana foto dalam ember Amazon S3 dapat dilihat. Lihat bagian induk untuk detail dan prasyarat.

HTML untuk contoh:

```
<!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>
```

Kode contoh ini dapat ditemukan di sini GitHub.

Kode skrip browser untuk contoh:

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

Kode contoh ini dapat ditemukan di sini GitHub.

Mengunggah Foto ke Amazon S3 dari Browser

Contoh kode skrip browser ini menunjukkan:

- Cara membuat aplikasi browser yang memungkinkan pengguna membuat album foto di bucket Amazon S3 dan mengunggah foto ke dalam album.

Skenario

Dalam contoh ini, halaman HTML sederhana menyediakan aplikasi berbasis browser untuk membuat album foto di ember Amazon S3 tempat Anda dapat mengunggah foto. Aplikasi ini memungkinkan Anda menghapus foto dan album yang Anda tambahkan.

Skrip browser menggunakan SDK JavaScript untuk berinteraksi dengan bucket Amazon S3. Gunakan metode berikut dari kelas klien Amazon S3 untuk mengaktifkan aplikasi album foto:

- `listObjects`

- `headObject`
- `putObject`
- `upload`
- `deleteObject`
- `deleteObjects`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Di konsol Amazon S3, buat bucket Amazon S3 yang akan Anda gunakan untuk menyimpan foto di album. Untuk informasi selengkapnya tentang membuat bucket di konsol, lihat Membuat Bucket di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon. Pastikan Anda memiliki izin Baca dan Tulis pada Objek. Untuk informasi selengkapnya tentang menyetel izin bucket, lihat Menyetel izin untuk akses situs web.
- Di konsol Amazon Cognito, buat kumpulan identitas Amazon Cognito menggunakan Identitas Federasi dengan akses diaktifkan untuk pengguna yang tidak diautentikasi di Wilayah yang sama dengan bucket Amazon S3. Anda perlu menyertakan ID kumpulan identitas dalam kode untuk mendapatkan kredensil skrip browser. Untuk informasi selengkapnya tentang Identitas Federasi Amazon Cognito, lihat Kumpulan Identitas Amazon Cognito (Identitas Federasi) di Panduan Pengembang Amazon Cognito.
- Di konsol IAM, temukan peran IAM yang dibuat oleh Amazon Cognito untuk pengguna yang tidak diautentikasi. Tambahkan kebijakan berikut untuk memberikan izin baca dan tulis ke bucket Amazon S3. Untuk informasi selengkapnya tentang membuat peran IAM, lihat Membuat Peran untuk Mendelegasikan Izin ke AWS Layanan di Panduan Pengguna IAM.

  Gunakan kebijakan peran ini untuk peran IAM yang dibuat oleh Amazon Cognito untuk pengguna yang tidak diautentikasi.

  > **Warning**
  >
  > Jika Anda mengaktifkan akses untuk pengguna yang tidak diautentikasi, Anda akan memberikan akses tulis ke bucket, dan semua objek di bucket, kepada siapa pun di dunia. Postur keamanan ini berguna dalam contoh ini untuk membuatnya tetap fokus pada tujuan

utama contoh. Namun, dalam banyak situasi langsung, keamanan yang lebih ketat, seperti menggunakan pengguna yang diautentikasi dan kepemilikan objek, sangat disarankan.

```
{
   "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/*"
         ]
      }
   ]
}
```

Mengonfigurasi CORS

Sebelum skrip browser dapat mengakses bucket Amazon S3, Anda harus terlebih dahulu mengatur konfigurasi CORS-nya sebagai berikut.

> **Important**
>
> Pada konsol S3 baru, konfigurasi CORS harus 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>
```

Halaman Web

HTML untuk aplikasi unggah foto terdiri dari elemen di mana skrip browser membuat antarmuka pengguna unggah. <script>Elemen pertama menambahkan SDK ke skrip browser. <script>Elemen kedua menambahkan JavaScript file eksternal yang memegang kode skrip 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>
```

Mengkonfigurasi SDK

Dapatkan kredensional yang diperlukan untuk mengonfigurasi SDK dengan memanggil `CognitoIdentityCredentials` metode, dengan menyediakan ID kumpulan identitas Amazon Cognito. Selanjutnya, buat objek `AWS.S3` layanan.

```
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 },
});
```

Hampir semua kode lainnya dalam contoh ini diatur ke dalam serangkaian fungsi yang mengumpulkan dan menyajikan informasi tentang album di ember, mengunggah dan menampilkan foto yang diunggah ke album, dan menghapus foto dan album. Fungsi-fungsi tersebut adalah:

- `listAlbums`
- `createAlbum`
- `viewAlbum`
- `addPhoto`
- `deleteAlbum`
- `deletePhoto`

Daftar Album di Bucket

Aplikasi ini membuat album di bucket Amazon S3 sebagai objek yang kuncinya dimulai dengan karakter garis miring, yang menunjukkan fungsi objek sebagai folder. Untuk mencantumkan semua album yang ada di bucket, `listAlbums` fungsi aplikasi memanggil `listObjects` metode objek `AWS.S3` layanan saat menggunakan `commonPrefix` sehingga panggilan hanya mengembalikan objek yang digunakan sebagai album.

Sisa fungsi mengambil daftar album dari bucket Amazon S3 dan menghasilkan HTML yang diperlukan untuk menampilkan daftar album di halaman web. Ini juga memungkinkan menghapus dan membuka album individu.

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

Membuat Album di Bucket

Untuk membuat album di bucket Amazon S3, `createAlbum` fungsi aplikasi pertama-tama memvalidasi nama yang diberikan untuk album baru untuk memastikannya berisi karakter yang sesuai. Fungsi tersebut kemudian membentuk kunci objek Amazon S3, meneruskannya ke `headObject` metode objek layanan Amazon S3. Metode ini mengembalikan metadata untuk kunci tertentu, jadi jika mengembalikan data, maka objek dengan kunci itu sudah ada.

Jika album belum ada, fungsi memanggil `putObject` metode objek `AWS.S3` layanan untuk membuat album. Kemudian memanggil `viewAlbum` fungsi untuk menampilkan album kosong baru.

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

Melihat Album

Untuk menampilkan konten album di bucket Amazon S3, `viewAlbum` fungsi aplikasi mengambil nama album dan membuat kunci Amazon S3 untuk album itu. Fungsi kemudian memanggil `listObjects` metode objek `AWS.S3` layanan untuk mendapatkan daftar semua objek (foto) dalam album.

Sisa fungsi mengambil daftar objek (foto) dari album dan menghasilkan HTML yang diperlukan untuk menampilkan foto di halaman web. Ini juga memungkinkan menghapus foto individu dan menavigasi kembali ke daftar 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);
  });
}
```

Menambahkan Foto ke Album

Untuk mengunggah foto ke album di bucket Amazon S3, `addPhoto` fungsi aplikasi menggunakan elemen pemilih file di halaman web untuk mengidentifikasi file yang akan diunggah. Kemudian membentuk kunci untuk foto untuk mengunggah dari nama album saat ini dan nama file.

Fungsi ini memanggil `upload` metode objek layanan Amazon S3 untuk mengunggah foto. Setelah mengunggah foto, fungsi menampilkan ulang album sehingga foto yang diunggah muncul.

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

Menghapus Foto

Untuk menghapus foto dari album di bucket Amazon S3, `deletePhoto` fungsi aplikasi memanggil `deleteObject` metode objek layanan Amazon S3. Ini menghapus foto yang ditentukan oleh `photoKey` nilai yang diteruskan ke fungsi.

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

Menghapus Album

Untuk menghapus album di bucket Amazon S3, `deleteAlbum` fungsi aplikasi memanggil `deleteObjects` metode objek layanan 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();
      }
    );
  });
```

```
}
```

Mengunggah Foto ke Amazon S3: Kode Lengkap

Bagian ini berisi HTML lengkap dan JavaScript kode untuk contoh di mana foto diunggah ke album foto Amazon S3. Lihat bagian induk untuk detail dan prasyarat.

HTML untuk contoh:

```
<!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>
```

Kode contoh ini dapat ditemukan di sini GitHub.

Kode skrip browser untuk contoh:

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

Kode contoh ini dapat ditemukan di sini GitHub.

Amazon S3 Node.js Contoh

Topik berikut menunjukkan contoh bagaimana AWS SDK untuk JavaScript dapat digunakan untuk berinteraksi dengan bucket Amazon S3 menggunakan Node.js.

Topik

- Membuat dan Menggunakan Bucket Amazon S3
- Mengkonfigurasi Bucket Amazon S3
- Mengelola Izin Akses Bucket Amazon S3
- Bekerja dengan Kebijakan Bucket Amazon S3
- Menggunakan Bucket Amazon S3 sebagai Host Web Statis

Membuat dan Menggunakan Bucket Amazon S3

Contoh kode Node.js ini menunjukkan:

- Cara mendapatkan dan menampilkan daftar bucket Amazon S3 di akun Anda.
- Cara membuat bucket Amazon S3.
- Cara mengunggah objek ke ember tertentu.

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mendapatkan daftar bucket Amazon S3 yang ada, membuat bucket, dan mengunggah file ke bucket tertentu. Modul Node.js ini menggunakan SDK JavaScript untuk mendapatkan informasi dari dan mengunggah file ke bucket Amazon S3 menggunakan metode kelas klien Amazon S3 berikut:

- `listBuckets`
- `createBucket`
- `listObjects`
- `upload`
- `deleteBucket`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mengkonfigurasi SDK

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global lalu menyetel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur ke`us-west-2`.

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

Menampilkan Daftar Bucket Amazon S3

Buat modul Node.js dengan nama files3_listbuckets.js. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon Simple Storage Service, buat objek AWS.S3 layanan. Panggil listBuckets metode objek layanan Amazon S3 untuk mengambil daftar bucket Anda. dataParameter fungsi callback memiliki Buckets properti yang berisi array peta untuk mewakili bucket. Tampilkan daftar ember dengan masuk ke konsol.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_listbuckets.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat sebuah Bucket Amazon S3

Buat modul Node.js dengan nama files3_createbucket.js. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat objek AWS.S3 layanan. Modul akan mengambil satu argumen baris perintah untuk menentukan nama bucket baru.

Tambahkan variabel untuk menahan parameter yang digunakan untuk memanggil createBucket metode objek layanan Amazon S3, termasuk nama untuk bucket yang baru dibuat. Fungsi callback mencatat lokasi bucket baru ke konsol setelah Amazon S3 berhasil membuatnya.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_createbucket.js BUCKET_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengunggah File ke Bucket Amazon S3

Buat modul Node.js dengan nama files3_upload.js. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat objek AWS.S3 layanan. Modul akan mengambil dua argumen baris perintah, yang pertama untuk menentukan bucket tujuan dan yang kedua untuk menentukan file yang akan diunggah.

Buat variabel dengan parameter yang diperlukan untuk memanggil upload metode objek layanan Amazon S3. Berikan nama bucket target dalam Bucket parameter. KeyParameter diatur ke nama file yang dipilih, yang dapat Anda peroleh menggunakan path modul Node.js. BodyParameter diatur ke isi file, yang dapat Anda peroleh menggunakan createReadStream dari fs modul Node.js.

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

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

// Create S3 service object
var s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// call S3 to retrieve upload file to specified bucket
var uploadParams = { Bucket: process.argv[2], Key: "", Body: "" };
var file = process.argv[3];

// Configure the file stream and obtain the upload parameters
var fs = require("fs");
var fileStream = fs.createReadStream(file);
fileStream.on("error", function (err) {
  console.log("File Error", err);
});
uploadParams.Body = fileStream;
var path = require("path");
uploadParams.Key = path.basename(file);

// call S3 to retrieve upload file to specified bucket
s3.upload(uploadParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  }
  if (data) {
    console.log("Upload Success", data.Location);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_upload.js BUCKET_NAME FILE_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Objek di Bucket Amazon S3

Buat modul Node.js dengan nama files3_listobjects.js. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat objek AWS.S3 layanan.

Tambahkan variabel untuk menahan parameter yang digunakan untuk memanggil listObjects metode objek layanan Amazon S3, termasuk nama bucket yang akan dibaca. Fungsi callback mencatat daftar objek (file) atau pesan kegagalan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_listobjects.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Bucket Amazon S3

Buat modul Node.js dengan nama files3_deletebucket.js. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat objek AWS.S3 layanan.

Tambahkan variabel untuk menahan parameter yang digunakan untuk memanggil createBucket metode objek layanan Amazon S3, termasuk nama bucket yang akan dihapus. Ember harus kosong untuk menghapusnya. Fungsi callback mencatat pesan sukses atau kegagalan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_deletebucket.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengkonfigurasi Bucket Amazon S3

Contoh kode Node.js ini menunjukkan:

- Cara mengonfigurasi izin berbagi sumber daya lintas asal (CORS) untuk bucket.

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mencantumkan bucket Amazon S3 Anda dan untuk mengonfigurasi CORS dan pencatatan bucket. Modul Node.js menggunakan SDK JavaScript untuk mengonfigurasi bucket Amazon S3 yang dipilih menggunakan metode kelas klien Amazon S3 berikut:

- `getBucketCors`

- `putBucketCors`

Untuk informasi selengkapnya tentang penggunaan konfigurasi CORS dengan bucket Amazon S3, lihat Cross-Origin Resource Sharing (CORS) di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mengkonfigurasi SDK

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global lalu menyetel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur ke`us-west-2`.

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

Mengambil Konfigurasi Bucket CORS

Buat modul Node.js dengan nama file`s3_getcors.js`. Modul akan mengambil satu argumen baris perintah untuk menentukan bucket yang konfigurasi CORS-nya Anda inginkan. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat objek `AWS.S3` layanan.

Satu-satunya parameter yang perlu Anda lewati adalah nama bucket yang dipilih saat memanggil `getBucketCors` metode. Jika bucket saat ini memiliki konfigurasi CORS, konfigurasi tersebut dikembalikan oleh Amazon S3 sebagai properti parameter `CORSRules` yang diteruskan ke fungsi callback. `data`

Jika bucket yang dipilih tidak memiliki konfigurasi CORS, informasi tersebut dikembalikan ke fungsi callback dalam parameter. `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));
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_getcors.js BUCKET_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menyetel Konfigurasi Bucket CORS

Buat modul Node.js dengan nama files3_setcors.js. Modul ini mengambil beberapa argumen baris perintah, yang pertama menentukan bucket yang konfigurasi CORS yang ingin Anda atur. Argumen tambahan menyebutkan metode HTTP (POST, GET, PUT, PATCH, DELETE, POST) yang ingin Anda izinkan untuk bucket. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek AWS.S3 layanan. Selanjutnya buat objek JSON untuk menahan nilai untuk konfigurasi CORS seperti yang dipersyaratkan oleh putBucketCors metode objek AWS.S3 layanan. "Authorization"Tentukan AllowedHeaders nilai dan "*" AllowedOrigins nilainya. Tetapkan nilai AllowedMethods sebagai array kosong pada awalnya.

Tentukan metode yang diizinkan sebagai parameter baris perintah ke modul Node.js, tambahkan masing-masing metode yang cocok dengan salah satu parameter. Tambahkan konfigurasi CORS

yang dihasilkan ke array konfigurasi yang terdapat dalam parameter. `CORSRules` Tentukan bucket yang ingin Anda konfigurasikan untuk CORS dalam `Bucket` parameter.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah termasuk satu atau lebih metode HTTP seperti yang ditunjukkan.

```
node s3_setcors.js BUCKET_NAME get put
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengelola Izin Akses Bucket Amazon S3

Contoh kode Node.js ini menunjukkan:

- Cara mengambil atau mengatur daftar kontrol akses untuk bucket Amazon S3.

Skenario

Dalam contoh ini, modul Node.js digunakan untuk menampilkan daftar kontrol akses bucket (ACL) untuk bucket yang dipilih dan menerapkan perubahan pada ACL untuk bucket yang dipilih. Modul Node.js menggunakan SDK for JavaScript untuk mengelola izin akses bucket Amazon S3 menggunakan metode kelas klien Amazon S3 berikut:

- `getBucketAcl`
- `putBucketAcl`

Untuk informasi selengkapnya tentang daftar kontrol akses untuk bucket Amazon S3, lihat Mengelola Akses dengan ACLs di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mengkonfigurasi SDK

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global lalu menyetel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur ke`us-west-2`.

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

Mengambil Daftar Kontrol Akses Bucket Saat Ini

Buat modul Node.js dengan nama file`s3_getbucketacl.js`. Modul akan mengambil satu argumen baris perintah untuk menentukan bucket yang konfigurasi ACL yang Anda inginkan. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek `AWS.S3` layanan. Satu-satunya parameter yang perlu Anda lewati adalah nama bucket yang dipilih saat memanggil `getBucketAcl` metode. Konfigurasi daftar kontrol akses saat ini dikembalikan oleh Amazon S3 dalam `data` parameter yang diteruskan ke fungsi 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_getbucketacl.js BUCKET_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Bekerja dengan Kebijakan Bucket Amazon S3

Contoh kode Node.js ini menunjukkan:

- Cara mengambil kebijakan bucket bucket Amazon S3.
- Cara menambahkan atau memperbarui kebijakan bucket bucket Amazon S3.
- Cara menghapus kebijakan bucket bucket Amazon S3.

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengambil, menetapkan, atau menghapus kebijakan bucket di bucket Amazon S3. Modul Node.js menggunakan SDK for JavaScript untuk mengonfigurasi kebijakan bucket Amazon S3 yang dipilih menggunakan metode kelas klien Amazon S3 berikut:

- `getBucketPolicy`
- `putBucketPolicy`
- `deleteBucketPolicy`

Untuk informasi selengkapnya tentang kebijakan bucket untuk bucket Amazon S3, lihat Menggunakan Kebijakan Bucket dan Kebijakan Pengguna di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mengkonfigurasi SDK

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global lalu menyetel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur ke`us-west-2`.

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

Mengambil Kebijakan Bucket Saat Ini

Buat modul Node.js dengan nama files3_getbucketpolicy.js. Modul ini mengambil satu argumen baris perintah yang menentukan bucket yang kebijakannya Anda inginkan. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek AWS.S3 layanan. Satu-satunya parameter yang perlu Anda lewati adalah nama bucket yang dipilih saat memanggil getBucketPolicy metode. Jika bucket saat ini memiliki kebijakan, kebijakan tersebut dikembalikan oleh Amazon S3 dalam data parameter yang diteruskan ke fungsi callback.

Jika bucket yang dipilih tidak memiliki kebijakan, informasi tersebut dikembalikan ke fungsi callback dalam error parameter.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_getbucketpolicy.js BUCKET_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menetapkan Kebijakan Bucket Sederhana

Buat modul Node.js dengan nama files3_setbucketpolicy.js. Modul ini mengambil satu argumen baris perintah yang menentukan bucket yang kebijakannya ingin Anda terapkan. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek AWS.S3 layanan. Kebijakan bucket ditentukan di JSON. Pertama, buat objek JSON yang berisi semua nilai untuk menentukan kebijakan kecuali Resource nilai yang mengidentifikasi bucket.

Format Resource string yang diperlukan oleh kebijakan, dengan memasukkan nama bucket yang dipilih. Masukkan string itu ke objek JSON. Siapkan parameter untuk putBucketPolicy metode, termasuk nama bucket dan kebijakan JSON yang dikonversi ke nilai string.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_setbucketpolicy.js BUCKET_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Kebijakan Bucket

Buat modul Node.js dengan nama files3_deletebucketpolicy.js. Modul ini mengambil satu argumen baris perintah yang menentukan bucket yang kebijakannya ingin Anda hapus. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek AWS.S3 layanan. Satu-satunya parameter yang perlu Anda lewati saat memanggil deleteBucketPolicy metode adalah nama bucket yang dipilih.

```
// 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);
  }
```

```
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_deletebucketpolicy.js BUCKET_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Bucket Amazon S3 sebagai Host Web Statis

Contoh kode Node.js ini menunjukkan:

- Cara mengatur bucket Amazon S3 sebagai host web statis.

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengonfigurasi bucket Anda untuk bertindak sebagai host web statis. Modul Node.js menggunakan SDK JavaScript untuk mengonfigurasi bucket Amazon S3 yang dipilih menggunakan metode kelas klien Amazon S3 berikut:

- `getBucketWebsite`
- `putBucketWebsite`
- `deleteBucketWebsite`

Untuk informasi selengkapnya tentang menggunakan bucket Amazon S3 sebagai host web statis, lihat Hosting Situs Web Statis di Amazon S3 di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mengkonfigurasi SDK

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global lalu menyetel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur ke`us-west-2`.

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

Mengambil Konfigurasi Situs Web Bucket Saat Ini

Buat modul Node.js dengan nama file`s3_getbucketwebsite.js`. Modul ini mengambil satu argumen baris perintah yang menentukan bucket yang konfigurasi situs webnya Anda inginkan. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek `AWS.S3` layanan. Buat fungsi yang mengambil konfigurasi situs web bucket saat ini untuk bucket yang dipilih dalam daftar keranjang. Satu-satunya parameter yang perlu Anda lewati adalah nama bucket yang dipilih saat memanggil `getBucketWebsite` metode. Jika bucket saat ini memiliki konfigurasi situs web, konfigurasi tersebut dikembalikan oleh Amazon S3 dalam `data` parameter yang diteruskan ke fungsi callback.

Jika bucket yang dipilih tidak memiliki konfigurasi situs web, informasi tersebut dikembalikan ke fungsi callback dalam `err` parameter.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_getbucketwebsite.js BUCKET_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menyetel Konfigurasi Situs Web Bucket

Buat modul Node.js dengan nama files3_setbucketwebsite.js. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat objek AWS.S3 layanan.

Buat fungsi yang menerapkan konfigurasi situs web bucket. Konfigurasi memungkinkan bucket yang dipilih berfungsi sebagai host web statis. Konfigurasi situs web ditentukan dalam JSON. Pertama, buat objek JSON yang berisi semua nilai untuk menentukan konfigurasi situs web, kecuali untuk Key nilai yang mengidentifikasi dokumen kesalahan, dan Suffix nilai yang mengidentifikasi dokumen indeks.

Masukkan nilai elemen input teks ke dalam objek JSON. Siapkan parameter untuk putBucketWebsite metode ini, termasuk nama bucket dan konfigurasi situs 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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_setbucketwebsite.js BUCKET_NAME INDEX_PAGE ERROR_PAGE
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Konfigurasi Situs Web Bucket

Buat modul Node.js dengan nama files3_deletebucketwebsite.js. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat objek AWS.S3 layanan.

Buat fungsi yang menghapus konfigurasi situs web untuk bucket yang dipilih. Satu-satunya parameter yang perlu Anda lewati saat memanggil deleteBucketWebsite metode adalah nama bucket yang dipilih.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node s3_deletebucketwebsite.js BUCKET_NAME
```

Kode contoh ini dapat ditemukan di sini GitHub.

Contoh Layanan Email Sederhana Amazon

Amazon Simple Email Service (Amazon SES) adalah layanan pengiriman email berbasis cloud yang dirancang untuk membantu pemasar digital dan pengembang aplikasi mengirim email pemasaran, pemberitahuan, dan transaksional. Ini adalah layanan yang andal dan hemat biaya untuk bisnis dari semua ukuran yang menggunakan email untuk tetap berhubungan dengan pelanggan mereka.

JavaScript API untuk Amazon SES diekspos melalui kelas `AWS.SES` klien. Untuk informasi selengkapnya tentang menggunakan kelas klien Amazon SES, lihat `Class: AWS.SES`di referensi API.

Topik

- Mengelola Identitas Amazon SES
- Bekerja dengan Template Email di Amazon SES
- Mengirim Email Menggunakan Amazon SES
- Menggunakan Filter Alamat IP untuk Tanda Terima Email di Amazon SES
- Menggunakan Aturan Tanda Terima di Amazon SES

Mengelola Identitas Amazon SES

Contoh kode Node.js ini menunjukkan:

- Cara memverifikasi alamat email dan domain yang digunakan dengan Amazon SES.
- Cara menetapkan kebijakan IAM ke identitas Amazon SES Anda.
- Cara membuat daftar semua identitas Amazon SES untuk AWS akun Anda.
- Cara menghapus identitas yang digunakan dengan Amazon SES.

Identitas Amazon SES adalah alamat email atau domain yang digunakan Amazon SES untuk mengirim email. Amazon SES mengharuskan Anda untuk memverifikasi identitas email Anda, mengonfirmasi bahwa Anda memilikinya dan mencegah orang lain menggunakannya.

Untuk detail tentang cara memverifikasi alamat email dan domain di Amazon SES, lihat Memverifikasi Alamat Email dan Domain di Amazon SES di Panduan Pengembang Layanan Email Sederhana Amazon. Untuk informasi tentang mengirim otorisasi di Amazon SES, lihat Ikhtisar Otorisasi Pengiriman Amazon SES.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk memverifikasi dan mengelola identitas Amazon SES. Modul Node.js menggunakan SDK JavaScript untuk memverifikasi alamat email dan domain, menggunakan metode kelas `AWS.SES` klien berikut:

- `listIdentities`

- `deleteIdentity`
- `verifyEmailIdentity`
- `verifyDomainIdentity`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file JSON kredensial, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mengkonfigurasi SDK

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global lalu setel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur ke`us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');

// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Daftar Identitas Anda

Dalam contoh ini, gunakan modul Node.js untuk mencantumkan alamat email dan domain yang akan digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_listidentities.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk melewati `IdentityType` dan parameter lainnya untuk `listIdentities` metode kelas `AWS.SES` klien. Untuk memanggil `listIdentities` metode, buat janji untuk menjalankan objek layanan Amazon SES, melewati objek parameter.

Kemudian tangani callback `response` in the promise. Yang `data` dikembalikan oleh janji berisi array identitas domain seperti yang ditentukan oleh `IdentityType` parameter.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ses_listidentities.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memverifikasi Identitas Alamat Email

Dalam contoh ini, gunakan modul Node.js untuk memverifikasi pengirim email yang akan digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_verifyemailidentity.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SES, buat objek `AWS.SES` layanan.

Buat objek untuk melewati `EmailAddress` parameter untuk `verifyEmailIdentity` metode kelas `AWS.SES` klien. Untuk memanggil `verifyEmailIdentity` metode ini, buat janji untuk menjalankan

objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Domain ditambahkan ke Amazon SES untuk diverifikasi.

```
node ses_verifyemailidentity.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memverifikasi Identitas Domain

Dalam contoh ini, gunakan modul Node.js untuk memverifikasi domain email yang akan digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_verifydomainidentity.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk melewati `Domain` parameter untuk `verifyDomainIdentity` metode kelas `AWS.SES` klien. Untuk memanggil `verifyDomainIdentity` metode, buat janji untuk menjalankan objek layanan Amazon SES, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Domain ditambahkan ke Amazon SES untuk diverifikasi.

```
node ses_verifydomainidentity.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Identitas

Dalam contoh ini, gunakan modul Node.js untuk menghapus alamat email atau domain yang digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_deleteidentity.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk melewati `Identity` parameter untuk `deleteIdentity` metode kelas `AWS.SES` klien. Untuk memanggil `deleteIdentity` metode, buat `request` untuk memanggil objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise..

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node ses_deleteidentity.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Bekerja dengan Template Email di Amazon SES

Contoh kode Node.js ini menunjukkan:

- Dapatkan daftar semua template email Anda.
- Mengambil dan memperbarui template email.
- Buat dan hapus template email.

Amazon SES memungkinkan Anda mengirim pesan email yang dipersonalisasi menggunakan templat email. Untuk detail tentang cara membuat dan menggunakan templat email di Amazon Simple Email Service, lihat Mengirim Email yang Dipersonalisasi Menggunakan Amazon SES API di Panduan Pengembang Layanan Email Sederhana Amazon.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk bekerja dengan template email. Modul Node.js menggunakan SDK JavaScript untuk membuat dan menggunakan templat email menggunakan metode kelas `AWS.SES` klien berikut:

- `listTemplates`
- `createTemplate`
- `getTemplate`
- `deleteTemplate`
- `updateTemplate`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang membuat file kredensial, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Daftar Template Email Anda

Dalam contoh ini, gunakan modul Node.js untuk membuat template email untuk digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_listtemplates.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk melewati parameter untuk `listTemplates` metode kelas `AWS.SES` klien. Untuk memanggil `listTemplates` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES mengembalikan daftar templat.

```
node ses_listtemplates.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mendapatkan Template Email

Dalam contoh ini, gunakan modul Node.js untuk mendapatkan template email untuk digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_gettemplate.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk melewati `TemplateName` parameter untuk `getTemplate` metode kelas `AWS.SES` klien. Untuk memanggil `getTemplate` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES mengembalikan detail template.

```
node ses_gettemplate.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat Template Email

Dalam contoh ini, gunakan modul Node.js untuk membuat template email untuk digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_createtemplate.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk melewati parameter untuk `createTemplate` metode kelas `AWS.SES` klien, termasuk,`TemplateName`, `HtmlPart``SubjectPart`, dan`TextPart`. Untuk memanggil `createTemplate` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Template ditambahkan ke Amazon SES.

```
node ses_createtemplate.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memperbarui Template Email

Dalam contoh ini, gunakan modul Node.js untuk membuat template email untuk digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_updatetemplate.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan nilai `Template` parameter yang ingin Anda perbarui di template, dengan `TemplateName` parameter yang diperlukan diteruskan ke `updateTemplate` metode kelas `AWS.SES` klien. Untuk memanggil `updateTemplate` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES mengembalikan detail template.

```
node ses_updatetemplate.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Template Email

Dalam contoh ini, gunakan modul Node.js untuk membuat template email untuk digunakan dengan Amazon SES. Buat modul Node.js dengan nama file`ses_deletetemplate.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan `TemplateName` parameter yang diperlukan ke `deleteTemplate` metode kelas `AWS.SES` klien. Untuk memanggil `deleteTemplate` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES mengembalikan detail template.

```
node ses_deletetemplate.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengirim Email Menggunakan Amazon SES

Contoh kode Node.js ini menunjukkan:

- Kirim teks atau email HTML.
- Kirim email berdasarkan template email.
- Kirim email massal berdasarkan template email.

Amazon SES API menyediakan dua cara berbeda bagi Anda untuk mengirim email, tergantung pada seberapa banyak kontrol yang Anda inginkan atas komposisi pesan email: diformat dan mentah. Untuk detailnya, lihat Mengirim Email Berformat Menggunakan Amazon SES API dan Mengirim Email Mentah Menggunakan Amazon SES API.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk mengirim email dengan berbagai cara. Modul Node.js menggunakan SDK JavaScript untuk membuat dan menggunakan templat email menggunakan metode kelas `AWS.SES` klien berikut:

- `sendEmail`
- `sendTemplatedEmail`
- `sendBulkTemplatedEmail`

Tugas Prasyarat

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file JSON kredensial, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Persyaratan Pengiriman Pesan Email

Amazon SES membuat pesan email dan segera mengantri untuk dikirim. Untuk mengirim email menggunakan `SES.sendEmail` metode ini, pesan Anda harus memenuhi persyaratan berikut:

- Anda harus mengirim pesan dari alamat email atau domain yang diverifikasi. Jika Anda mencoba mengirim email menggunakan alamat atau domain yang tidak diverifikasi, operasi akan menghasilkan `"Email address not verified"` kesalahan.
- Jika akun Anda masih dalam kotak pasir Amazon SES, Anda hanya dapat mengirim ke alamat atau domain terverifikasi, atau ke alamat email yang terkait dengan Simulator Kotak Surat Amazon SES. Untuk informasi selengkapnya, lihat Memverifikasi Alamat Email dan Domain di Panduan Pengembang Layanan Email Sederhana Amazon.
- Ukuran total pesan, termasuk lampiran, harus lebih kecil dari 10 MB.
- Pesan harus menyertakan setidaknya satu alamat email penerima. Alamat penerima dapat berupa alamat Kepada:, alamat CC:, atau alamat BCC:. Jika alamat email penerima tidak valid (artinya, tidak dalam format`UserName@[SubDomain.]Domain.TopLevelDomain`), seluruh pesan ditolak, bahkan jika pesan berisi penerima lain yang valid.
- Pesan tidak dapat menyertakan lebih dari 50 penerima, di bidang Kepada:, CC: dan BCC:. Jika Anda perlu mengirim pesan email ke audiens yang lebih besar, Anda dapat membagi daftar

penerima Anda menjadi grup 50 atau kurang, dan kemudian memanggil `sendEmail` metode beberapa kali untuk mengirim pesan ke setiap grup.

Mengirim Email

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_sendemail.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan nilai parameter yang menentukan email yang akan dikirim, termasuk alamat pengirim dan penerima, subjek, badan email dalam format teks biasa dan HTML, ke `sendEmail` metode kelas `AWS.SES` klien. Untuk memanggil `sendEmail` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Email diantrian untuk dikirim oleh Amazon SES.

```
node ses_sendemail.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengirim Email Menggunakan Template

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_sendtemplatedemail.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan nilai parameter yang menentukan email yang akan dikirim, termasuk alamat pengirim dan penerima, subjek, badan email dalam format teks biasa dan HTML, ke `sendTemplatedEmail` metode kelas `AWS.SES` klien. Untuk memanggil `sendTemplatedEmail` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

```
// 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);
```

```
  });
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Email diantrian untuk dikirim oleh Amazon SES.

```
node ses_sendtemplatedemail.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengirim Email Massal Menggunakan Template

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_sendbulktemplatedemail.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan nilai parameter yang menentukan email yang akan dikirim, termasuk alamat pengirim dan penerima, subjek, badan email dalam format teks biasa dan HTML, ke `sendBulkTemplatedEmail` metode kelas `AWS.SES` klien. Untuk memanggil `sendBulkTemplatedEmail` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Email diantrian untuk dikirim oleh Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Filter Alamat IP untuk Tanda Terima Email di Amazon SES

Contoh kode Node.js ini menunjukkan:

- Buat filter alamat IP untuk menerima atau menolak email yang berasal dari alamat IP atau rentang alamat IP.

- Buat daftar filter alamat IP Anda saat ini.
- Hapus filter alamat IP.

Di Amazon SES, filter adalah struktur data yang terdiri dari nama, rentang alamat IP, dan apakah akan mengizinkan atau memblokir email darinya. Alamat IP yang ingin Anda blokir atau izinkan ditentukan sebagai alamat IP tunggal atau rentang alamat IP dalam notasi Classless Inter-Domain Routing (CIDR). Untuk detail tentang cara Amazon SES menerima email, lihat Konsep Penerimaan Email Amazon SES di Panduan Pengembang Layanan Email Sederhana Amazon.

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengirim email dalam berbagai cara. Modul Node.js menggunakan SDK JavaScript untuk membuat dan menggunakan templat email menggunakan metode kelas `AWS.SES` klien berikut:

- `createReceiptFilter`
- `listReceiptFilters`
- `deleteReceiptFilter`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mengkonfigurasi SDK

Konfigurasikan SDK untuk JavaScript dengan membuat objek konfigurasi global lalu setel Wilayah untuk kode Anda. Dalam contoh ini, Region diatur ke`us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
```

```
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Membuat Filter Alamat IP

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_createreceiptfilter.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan nilai parameter yang menentukan filter IP, termasuk nama filter, alamat IP atau rentang alamat untuk difilter, dan apakah akan mengizinkan atau memblokir lalu lintas email dari alamat yang difilter. Untuk memanggil `createReceiptFilter` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Filter dibuat di Amazon SES.

```
node ses_createreceiptfilter.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Cantumkan Filter Alamat IP Anda

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_listreceiptfilters.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek parameter kosong. Untuk memanggil `listReceiptFilters` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES mengembalikan daftar filter.

```
node ses_listreceiptfilters.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Filter Alamat IP

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_deletereceiptfilter.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan nama filter IP untuk dihapus. Untuk memanggil `deleteReceiptFilter` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Filter dihapus dari Amazon SES.

```
node ses_deletereceiptfilter.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Aturan Tanda Terima di Amazon SES

Contoh kode Node.js ini menunjukkan:

- Buat dan hapus aturan tanda terima.
- Atur aturan tanda terima ke dalam set aturan penerimaan.

Aturan tanda terima di Amazon SES menentukan apa yang harus dilakukan dengan email yang diterima untuk alamat email atau domain yang Anda miliki. Aturan tanda terima berisi kondisi dan daftar tindakan yang diurutkan. Jika penerima email masuk cocok dengan penerima yang ditentukan dalam kondisi aturan tanda terima, Amazon SES akan melakukan tindakan yang ditentukan oleh aturan tanda terima.

Untuk menggunakan Amazon SES sebagai penerima email Anda, Anda harus memiliki setidaknya satu aturan tanda terima aktif yang ditetapkan. Kumpulan aturan tanda terima adalah kumpulan aturan tanda terima yang diurutkan yang menentukan apa yang harus dilakukan Amazon SES dengan email yang diterimanya di seluruh domain terverifikasi Anda. Untuk informasi selengkapnya, lihat Membuat Aturan Tanda Terima untuk Menerima Email Amazon SES dan Membuat Aturan Tanda Terima yang Ditetapkan untuk Penerimaan Email Amazon SES di Panduan Pengembang Layanan Email Sederhana Amazon.

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengirim email dalam berbagai cara. Modul Node.js menggunakan SDK JavaScript untuk membuat dan menggunakan templat email menggunakan metode kelas `AWS.SES` klien berikut:

- `createReceiptRule`
- `deleteReceiptRule`
- `createReceiptRuleSet`
- `deleteReceiptRuleSet`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file JSON kredensial, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Membuat Aturan Tanda Terima Amazon S3

Setiap aturan tanda terima untuk Amazon SES berisi daftar tindakan yang diurutkan. Contoh ini membuat aturan tanda terima dengan tindakan Amazon S3, yang mengirimkan pesan email ke bucket Amazon S3. Untuk detail tentang tindakan aturan penerimaan, lihat Opsi Tindakan di Panduan Pengembang Layanan Email Sederhana Amazon.

Agar Amazon SES dapat menulis email ke bucket Amazon S3, buat kebijakan bucket yang memberikan `PutObject` izin ke Amazon SES. Untuk informasi tentang cara membuat kebijakan bucket ini, lihat Berikan Izin Amazon SES untuk Menulis ke Bucket Amazon S3 Anda di Panduan Pengembang Layanan Email Sederhana Amazon.

Dalam contoh ini, gunakan modul Node.js untuk membuat aturan tanda terima di Amazon SES untuk menyimpan pesan yang diterima di bucket Amazon S3. Buat modul Node.js dengan nama file`ses_createreceiptrule.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek parameter untuk meneruskan nilai yang diperlukan untuk membuat set aturan penerimaan. Untuk memanggil `createReceiptRuleSet` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES membuat aturan tanda terima.

```
node ses_createreceiptrule.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Aturan Tanda Terima

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_deletereceiptrule.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek parameter untuk meneruskan nama untuk menghapus aturan tanda terima. Untuk memanggil `deleteReceiptRule` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES membuat daftar set aturan tanda terima.

```
node ses_deletereceiptrule.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat Set Aturan Tanda Terima

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_createreceiptruleset.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek parameter untuk meneruskan nama untuk set aturan penerimaan baru. Untuk memanggil `createReceiptRuleSet` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES membuat daftar set aturan tanda terima.

```
node ses_createreceiptruleset.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Set Aturan Tanda Terima

Dalam contoh ini, gunakan modul Node.js untuk mengirim email dengan Amazon SES. Buat modul Node.js dengan nama file`ses_deletereceiptruleset.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan nama untuk aturan tanda terima yang ditetapkan untuk dihapus. Untuk memanggil `deleeReceiptRuleSet` metode ini, buat janji untuk menjalankan objek layanan Amazon SES, melewati parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah. Amazon SES membuat daftar set aturan tanda terima.

```
node ses_deletereceiptruleset.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Contoh Layanan Pemberitahuan Sederhana Amazon

Amazon Simple Notification Service (Amazon SNS) adalah layanan web yang mengoordinasikan dan mengelola pengiriman atau pengiriman pesan untuk berlangganan titik akhir atau klien.

Di Amazon SNS, ada dua jenis klien—penerbit dan pelanggan—juga disebut sebagai produsen dan konsumen.

Penerbit berkomunikasi secara asinkron dengan pelanggan dengan memproduksi dan mengirim pesan ke suatu topik, yang merupakan jalur akses logis dan saluran komunikasi. Pelanggan (server web, alamat email, antrian Amazon SQS, fungsi Lambda) mengkonsumsi atau menerima pesan atau pemberitahuan melalui salah satu protokol yang didukung (Amazon SQS, HTTP/S, email, SMS,) ketika mereka berlangganan topik. AWS Lambda

JavaScript API untuk Amazon SNS diekspos melalui file. `Class: AWS.SNS`

Topik

- Mengelola Topik di Amazon SNS
- Menerbitkan Pesan di Amazon SNS
- Mengelola Langganan di Amazon SNS
- Mengirim Pesan SMS dengan Amazon SNS

Mengelola Topik di Amazon SNS

Contoh kode Node.js ini menunjukkan:

- Cara membuat topik di Amazon SNS tempat Anda dapat mempublikasikan notifikasi.
- Cara menghapus topik yang dibuat di Amazon SNS.
- Cara mendapatkan daftar topik yang tersedia.

- Cara mendapatkan dan mengatur atribut topik.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk membuat, membuat daftar, dan menghapus topik Amazon SNS, dan untuk menangani atribut topik. Modul Node.js menggunakan SDK JavaScript untuk mengelola topik menggunakan metode kelas `AWS.SNS` klien berikut:

- `createTopic`
- `listTopics`
- `deleteTopic`
- `getTopicAttributes`
- `setTopicAttributes`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file JSON kredensial, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Membuat Topik

Dalam contoh ini, gunakan modul Node.js untuk membuat topik Amazon SNS. Buat modul Node.js dengan nama file`sns_createtopic.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek untuk meneruskan topik baru ke `createTopic` metode kelas `AWS.SNS` klien. `Name` Untuk memanggil `createTopic` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise. Yang `data` dikembalikan oleh janji berisi ARN dari topik tersebut.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_createtopic.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Topik Anda

Dalam contoh ini, gunakan modul Node.js untuk mencantumkan semua topik Amazon SNS. Buat modul Node.js dengan nama file`sns_listtopics.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek kosong untuk diteruskan ke `listTopics` metode kelas `AWS.SNS` klien. Untuk memanggil `listTopics` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise. Yang `data` dikembalikan oleh janji berisi berbagai topik Anda 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);
  });
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_listtopics.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Topik

Dalam contoh ini, gunakan modul Node.js untuk menghapus topik Amazon SNS. Buat modul Node.js dengan nama file`sns_deletetopic.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang `TopicArn` berisi topik yang akan dihapus untuk diteruskan ke `deleteTopic` metode kelas `AWS.SNS` klien. Untuk memanggil `deleteTopic` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_deletetopic.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mendapatkan Atribut Topik

Dalam contoh ini, gunakan modul Node.js untuk mengambil atribut topik Amazon SNS. Buat modul Node.js dengan nama file`sns_gettopicattributes.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang `TopicArn` berisi topik yang akan dihapus untuk diteruskan ke `getTopicAttributes` metode kelas `AWS.SNS` klien. Untuk memanggil `getTopicAttributes` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_gettopicattributes.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Pengaturan Atribut Topik

Dalam contoh ini, gunakan modul Node.js untuk menyetel atribut yang dapat berubah dari topik Amazon SNS. Buat modul Node.js dengan nama file`sns_settopicattributes.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi parameter untuk pembaruan atribut, termasuk topik yang atributnya ingin Anda tetapkan, nama atribut yang akan ditetapkan, dan nilai baru untuk atribut tersebut. `TopicArn` Anda hanya dapat mengatur`Policy`,`DisplayName`, dan `DeliveryPolicy` atribut. Lewati parameter ke `setTopicAttributes` metode kelas `AWS.SNS` klien. Untuk memanggil `setTopicAttributes` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_settopicattributes.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menerbitkan Pesan di Amazon SNS

Contoh kode Node.js ini menunjukkan:

- Cara mempublikasikan pesan ke topik Amazon SNS.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk mempublikasikan pesan dari Amazon SNS ke titik akhir topik, email, atau nomor telepon. Modul Node.js menggunakan SDK JavaScript untuk mengirim pesan menggunakan metode kelas `AWS.SNS` klien ini:

- `publish`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file JSON kredensial, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Menerbitkan Pesan ke Topik Amazon SNS

Dalam contoh ini, gunakan modul Node.js untuk mempublikasikan pesan ke topik Amazon SNS. Buat modul Node.js dengan nama file`sns_publishtotopic.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi parameter untuk menerbitkan pesan, termasuk teks pesan dan ARN dari topik Amazon SNS. Untuk detail tentang atribut SMS yang tersedia, lihat Mengatur SMSAttributes.

Lewati parameter ke `publish` metode kelas `AWS.SNS` klien. Buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani respons dalam panggilan balik janji.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_publishtotopic.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengelola Langganan di Amazon SNS

Contoh kode Node.js ini menunjukkan:

- Cara membuat daftar semua langganan ke topik Amazon SNS.
- Cara berlangganan alamat email, titik akhir aplikasi, atau AWS Lambda fungsi ke topik Amazon SNS.
- Cara berhenti berlangganan dari topik Amazon SNS.

Skenario

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk mempublikasikan pesan notifikasi ke topik Amazon SNS. Modul Node.js menggunakan SDK JavaScript untuk mengelola topik menggunakan metode kelas `AWS.SNS` klien berikut:

- `subscribe`
- `confirmSubscription`
- `listSubscriptionsByTopic`
- `unsubscribe`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file JSON kredensial, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Daftar Langganan ke Topik

Dalam contoh ini, gunakan modul Node.js untuk mencantumkan semua langganan ke topik Amazon SNS. Buat modul Node.js dengan nama file`sns_listsubscriptions.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi `TopicArn` parameter untuk topik yang langganannya ingin Anda daftarkan. Lewati parameter ke `listSubscriptionsByTopic` metode kelas `AWS.SNS` klien. Untuk memanggil `listSubscriptionsByTopic` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_listsubscriptions.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Berlangganan Alamat Email ke Topik

Dalam contoh ini, gunakan modul Node.js untuk berlangganan alamat email sehingga menerima pesan email SMTP dari topik Amazon SNS. Buat modul Node.js dengan nama file`sns_subscribeemail.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi `Protocol` parameter untuk menentukan `email` protokol, topik `TopicArn` untuk berlangganan, dan alamat email sebagai pesan`Endpoint`. Lewati parameter ke `subscribe` metode kelas `AWS.SNS` klien. Anda dapat menggunakan `subscribe` metode ini untuk berlangganan beberapa titik akhir yang berbeda ke topik Amazon SNS, tergantung pada nilai yang digunakan untuk parameter yang diteruskan, seperti contoh lain dalam topik ini akan ditampilkan.

Untuk memanggil `subscribe` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_subscribeemail.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Berlangganan Endpoint Aplikasi ke Topik

Dalam contoh ini, gunakan modul Node.js untuk berlangganan titik akhir aplikasi seluler sehingga menerima pemberitahuan dari topik Amazon SNS. Buat modul Node.js dengan nama file`sns_subscribeapp.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi `Protocol` parameter untuk menentukan `application` protokol, topik `TopicArn` untuk berlangganan, dan ARN dari titik akhir aplikasi seluler untuk parameter tersebut. `Endpoint` Lewati parameter ke `subscribe` metode kelas `AWS.SNS` klien.

Untuk memanggil `subscribe` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_subscribeapp.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Berlangganan Fungsi Lambda ke Topik

Dalam contoh ini, gunakan modul Node.js untuk berlangganan suatu AWS Lambda fungsi sehingga menerima pemberitahuan dari topik Amazon SNS. Buat modul Node.js dengan nama file`sns_subscribelambda.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi `Protocol` parameter, tentukan `lambda` protokol, topik `TopicArn` untuk berlangganan, dan ARN fungsi AWS Lambda sebagai `Endpoint` parameter. Lewati parameter ke `subscribe` metode kelas `AWS.SNS` klien.

Untuk memanggil `subscribe` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_subscribelambda.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Berhenti berlangganan dari topik

Dalam contoh ini, gunakan modul Node.js untuk berhenti berlangganan langganan topik Amazon SNS. Buat modul Node.js dengan nama file`sns_unsubscribe.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi `SubscriptionArn` parameter, tentukan ARN langganan untuk berhenti berlangganan. Lewati parameter ke `unsubscribe` metode kelas `AWS.SNS` klien.

Untuk memanggil `unsubscribe` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_unsubscribe.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengirim Pesan SMS dengan Amazon SNS

Contoh kode Node.js ini menunjukkan:

- Cara mendapatkan dan mengatur preferensi pesan SMS untuk Amazon SNS.
- Cara memeriksa nomor telepon untuk melihat apakah telah memilih untuk tidak menerima pesan SMS.
- Cara mendapatkan daftar nomor telepon yang telah memilih untuk tidak menerima pesan SMS.
- Cara mengirim pesan SMS.

Skenario

Anda dapat menggunakan Amazon SNS untuk mengirim pesan teks, atau pesan SMS, ke perangkat yang mendukung SMS. Anda dapat mengirim pesan langsung ke sebuah nomor telepon, atau Anda dapat mengirim pesan ke beberapa nomor telepon sekaligus dengan berlangganan topik untuk nomor telepon tersebut dan mengirim pesan Anda ke topik tersebut.

Dalam contoh ini, Anda menggunakan serangkaian modul Node.js untuk mempublikasikan pesan teks SMS dari Amazon SNS ke perangkat berkemampuan SMS. Modul Node.js menggunakan SDK JavaScript untuk mempublikasikan pesan SMS menggunakan metode kelas `AWS.SNS` klien berikut:

- `getSMSAttributes`
- `setSMSAttributes`

- `checkIfPhoneNumberIsOptedOut`
- `listPhoneNumbersOptedOut`
- `publish`

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file JSON kredensial, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mendapatkan Atribut SMS

Gunakan Amazon SNS untuk menentukan preferensi untuk pesan SMS, seperti bagaimana pengiriman Anda dioptimalkan (untuk biaya atau untuk pengiriman yang andal), batas pengeluaran bulanan Anda, cara pengiriman pesan dicatat, dan apakah akan berlangganan laporan penggunaan SMS harian. Preferensi ini diambil dan ditetapkan sebagai atribut SMS untuk Amazon SNS.

Dalam contoh ini, gunakan modul Node.js untuk mendapatkan atribut SMS saat ini di Amazon SNS. Buat modul Node.js dengan nama file`sns_getsmstype.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya. Buat objek yang berisi parameter untuk mendapatkan atribut SMS, termasuk nama-nama atribut individual yang akan didapat. Untuk detail tentang atribut SMS yang tersedia, lihat Mengatur SMSAttributes Referensi API Layanan Pemberitahuan Sederhana Amazon.

Contoh ini mendapatkan `DefaultSMSType` atribut, yang mengontrol apakah pesan SMS dikirim sebagai`Promotional`, yang mengoptimalkan pengiriman pesan untuk menimbulkan biaya terendah, atau as`Transactional`, yang mengoptimalkan pengiriman pesan untuk mencapai keandalan tertinggi. Lewati parameter ke `setTopicAttributes` metode kelas `AWS.SNS` klien. Untuk memanggil `getSMSAttributes` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_getsmstype.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengatur Atribut SMS

Dalam contoh ini, gunakan modul Node.js untuk mendapatkan atribut SMS saat ini di Amazon SNS. Buat modul Node.js dengan nama file`sns_setsmstype.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya. Buat objek yang berisi parameter untuk mengatur atribut SMS, termasuk nama atribut individual yang akan ditetapkan dan nilai yang akan ditetapkan untuk masing-masing. Untuk detail tentang atribut SMS yang tersedia, lihat Mengatur SMSAttributes Referensi API Layanan Pemberitahuan Sederhana Amazon.

Contoh ini menetapkan `DefaultSMSType` atribut ke`Transactional`, yang mengoptimalkan pengiriman pesan untuk mencapai keandalan tertinggi. Lewati parameter ke `setTopicAttributes`

metode kelas `AWS.SNS` klien. Untuk memanggil `getSMSAttributes` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_setsmstype.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Memeriksa Apakah Nomor Telepon Telah Memilih Keluar

Dalam contoh ini, gunakan modul Node.js untuk memeriksa nomor telepon untuk melihat apakah telah memilih keluar dari menerima pesan SMS. Buat modul Node.js dengan nama

file`sns_checkphoneoptout.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya. Buat objek yang berisi nomor telepon untuk diperiksa sebagai parameter.

Contoh ini menetapkan `PhoneNumber` parameter untuk menentukan nomor telepon yang akan diperiksa. Lewati objek ke `checkIfPhoneNumberIsOptedOut` metode kelas `AWS.SNS` klien. Untuk memanggil `checkIfPhoneNumberIsOptedOut` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_checkphoneoptout.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Daftar Nomor Telepon yang Dipilih Keluar

Dalam contoh ini, gunakan modul Node.js untuk mendapatkan daftar nomor telepon yang telah memilih keluar dari menerima pesan SMS. Buat modul Node.js dengan nama file`sns_listnumbersoptedout.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya. Buat objek kosong sebagai parameter.

Lewati objek ke `listPhoneNumbersOptedOut` metode kelas `AWS.SNS` klien. Untuk memanggil `listPhoneNumbersOptedOut` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_listnumbersoptedout.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menerbitkan Pesan SMS

Dalam contoh ini, gunakan modul Node.js untuk mengirim pesan SMS ke nomor telepon. Buat modul Node.js dengan nama file`sns_publishsms.js`. Konfigurasikan SDK seperti yang ditunjukkan sebelumnya. Buat objek yang berisi `PhoneNumber` parameter `Message` dan.

Saat Anda mengirim pesan SMS, tentukan nomor telepon menggunakan format E.164. E.164 adalah standar untuk struktur nomor telepon yang digunakan untuk telekomunikasi internasional. Nomor telepon yang mengikuti format ini dapat memiliki maksimum 15 digit, dan diawali dengan karakter plus (+) dan kode negara. Misalnya, nomor telepon AS dalam format E.164 akan muncul sebagai XXX555 +1001 0100.

Contoh ini menetapkan `PhoneNumber` parameter untuk menentukan nomor telepon untuk mengirim pesan. Lewati objek ke `publish` metode kelas `AWS.SNS` klien. Untuk memanggil `publish` metode ini, buat janji untuk memanggil objek layanan Amazon SNS, melewati objek parameter. Kemudian tangani callback `response` in the promise.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sns_publishsms.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Amazon SQS Contoh

Amazon Simple Queue Service (Amazon SQS) adalah layanan antrian pesan yang cepat, andal, dapat diskalakan, dan dikelola sepenuhnya. Amazon SQS memungkinkan Anda memisahkan

komponen aplikasi cloud. Amazon SQS mencakup antrian standar dengan throughput dan at-least-once pemrosesan tinggi, dan antrian FIFO yang menyediakan pengiriman FIFO (first-in, first-out) dan pemrosesan tepat sekali.

JavaScript API untuk Amazon SQS diekspos melalui kelas `AWS.SQS` klien. Untuk informasi selengkapnya tentang menggunakan kelas klien Amazon SQS, lihat `Class: AWS.SQS`di referensi API.

Topik

- Menggunakan Antrian di Amazon SQS
- Mengirim dan Menerima Pesan di Amazon SQS
- Mengelola Batas Waktu Visibilitas di Amazon SQS
- Mengaktifkan Polling Panjang di Amazon SQS
- Menggunakan Antrian Surat Mati di Amazon SQS

Menggunakan Antrian di Amazon SQS

Contoh kode Node.js ini menunjukkan:

- Cara mendapatkan daftar semua antrian pesan Anda
- Cara mendapatkan URL untuk antrian tertentu
- Cara membuat dan menghapus antrian

Tentang Contoh

Dalam contoh ini, serangkaian modul Node.js digunakan untuk bekerja dengan antrian. Modul Node.js menggunakan SDK JavaScript untuk mengaktifkan antrian untuk memanggil metode berikut dari kelas klien: `AWS.SQS`

- `listQueues`
- `createQueue`
- `getQueueUrl`
- `deleteQueue`

Untuk informasi selengkapnya tentang pesan Amazon SQS, lihat Cara Kerja Antrian di Panduan Pengembang Layanan Antrian Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Daftar Antrian Anda

Buat modul Node.js dengan nama file`sqs_listqueues.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SQS, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk daftar antrian Anda, yang secara default adalah objek kosong. Panggil `listQueues` metode untuk mengambil daftar antrian. Callback mengembalikan semua antrian. URLs

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_listqueues.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Membuat Antrian

Buat modul Node.js dengan nama file`sqs_createqueue.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SQS, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk membuat daftar antrian Anda, yang harus menyertakan nama untuk antrian yang dibuat. Parameter juga dapat berisi atribut untuk antrian, seperti jumlah detik pengiriman pesan tertunda atau jumlah detik untuk menyimpan pesan yang diterima. Panggil metode `createQueue`. Callback mengembalikan URL antrian yang dibuat.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_createqueue.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mendapatkan URL untuk Antrian

Buat modul Node.js dengan nama file`sqs_getqueueurl.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SQS, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk membuat daftar antrian Anda, yang harus menyertakan nama antrian yang URL-nya Anda inginkan. Panggil metode `getQueueUrl`. Callback mengembalikan URL antrian yang ditentukan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_getqueueurl.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menghapus Antrian

Buat modul Node.js dengan nama file`sqs_deletequeue.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SQS, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus antrian, yang terdiri dari URL antrian yang ingin Anda hapus. Panggil metode `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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_deletequeue.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengirim dan Menerima Pesan di Amazon SQS

Contoh kode Node.js ini menunjukkan:

- Cara mengirim pesan dalam antrian.
- Cara menerima pesan dalam antrian.
- Cara menghapus pesan dalam antrian.

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengirim dan menerima pesan. Modul Node.js menggunakan SDK for JavaScript untuk mengirim dan menerima pesan dengan menggunakan metode kelas `AWS.SQS` klien berikut:

- `sendMessage`
- `receiveMessage`
- `deleteMessage`

Untuk informasi selengkapnya tentang pesan Amazon SQS, lihat Mengirim Pesan ke Antrian Amazon SQS dan Menerima serta Menghapus Pesan dari Antrian Amazon SQS di Panduan Pengembang Layanan Antrian Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

- Membuat antrean Amazon SQS. Untuk contoh membuat antrian, lihatMenggunakan Antrian di Amazon SQS.

Mengirim Pesan ke Antrian

Buat modul Node.js dengan nama filesqs_sendmessage.js. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SQS, buat objek AWS.SQS layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk pesan Anda, termasuk URL antrian yang ingin Anda kirimi pesan ini. Dalam contoh ini, pesan memberikan rincian tentang buku pada daftar buku fiksi terlaris termasuk judul, penulis, dan jumlah minggu dalam daftar.

Panggil metode sendMessage. Callback mengembalikan ID unik pesan.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_sendmessage.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menerima dan Menghapus Pesan dari Antrian

Buat modul Node.js dengan nama file`sqs_receivemessage.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SQS, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk pesan Anda, termasuk URL antrian dari mana Anda ingin menerima pesan. Dalam contoh ini, parameter menentukan penerimaan semua atribut pesan, serta penerimaan tidak lebih dari 10 pesan.

Panggil metode `receiveMessage`. Callback mengembalikan array `Message` objek dari mana Anda dapat mengambil `ReceiptHandle` untuk setiap pesan yang Anda gunakan untuk kemudian menghapus pesan itu. Buat objek JSON lain yang berisi parameter yang diperlukan untuk menghapus pesan, yang merupakan URL antrian dan nilainya. `ReceiptHandle` Panggil `deleteMessage` metode untuk menghapus pesan yang Anda terima.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_receivemessage.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengelola Batas Waktu Visibilitas di Amazon SQS

Contoh kode Node.js ini menunjukkan:

- Cara menentukan interval waktu di mana pesan yang diterima oleh antrian tidak terlihat.

Skenario

Dalam contoh ini, modul Node.js digunakan untuk mengelola batas waktu visibilitas. Modul Node.js menggunakan SDK for JavaScript untuk mengelola batas waktu visibilitas dengan menggunakan metode kelas klien ini: `AWS.SQS`

- `changeMessageVisibility`

Untuk informasi selengkapnya tentang batas waktu visibilitas Amazon SQS, lihat Batas Waktu Visibilitas di Panduan Pengembang Layanan Antrian Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama
- Membuat antrean Amazon SQS. Untuk contoh membuat antrian, lihatMenggunakan Antrian di Amazon SQS.
- Kirim pesan ke antrian. Untuk contoh pengiriman pesan ke antrian, lihatMengirim dan Menerima Pesan di Amazon SQS.

Mengubah Batas Waktu Visibilitas

Buat modul Node.js dengan nama file`sqs_changingvisibility.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon Simple Queue Service, buat objek `AWS.SQS` layanan. Menerima pesan dari antrean.

Setelah menerima pesan dari antrian, buat objek JSON yang berisi parameter yang diperlukan untuk mengatur batas waktu, termasuk URL antrian yang berisi pesan, yang `ReceiptHandle` dikembalikan saat pesan diterima, dan batas waktu baru dalam hitungan detik. Panggil metode `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");
    }
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_changingvisibility.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengaktifkan Polling Panjang di Amazon SQS

Contoh kode Node.js ini menunjukkan:

- Cara mengaktifkan polling panjang untuk antrian yang baru dibuat
- Cara mengaktifkan polling panjang untuk antrian yang ada
- Cara mengaktifkan polling panjang setelah menerima pesan

Skenario

Polling panjang mengurangi jumlah respons kosong dengan mengizinkan Amazon SQS menunggu waktu tertentu agar pesan tersedia dalam antrian sebelum mengirim respons. Selain itu, polling panjang menghilangkan respons kosong palsu dengan menanyakan semua server alih-alih pengambilan sampel server. Untuk mengaktifkan polling panjang, Anda harus menentukan waktu tunggu bukan nol untuk pesan yang diterima. Anda dapat melakukan ini dengan mengatur `ReceiveMessageWaitTimeSeconds` parameter antrian atau dengan mengatur `WaitTimeSeconds` parameter pada pesan saat diterima.

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengaktifkan polling panjang. Modul Node.js menggunakan SDK JavaScript untuk mengaktifkan polling panjang menggunakan metode kelas `AWS.SQS` klien berikut:

- `setQueueAttributes`
- `receiveMessage`
- `createQueue`

Untuk informasi selengkapnya tentang polling panjang Amazon SQS, lihat Polling Panjang di Panduan Pengembang Layanan Antrian Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

Mengaktifkan Polling Panjang Saat Membuat Antrian

Buat modul Node.js dengan nama file`sqs_longpolling_createqueue.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SQS, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk membuat antrian, termasuk nilai bukan nol untuk parameter. `ReceiveMessageWaitTimeSeconds` Panggil metode `createQueue`. Polling panjang kemudian diaktifkan untuk antrian.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_longpolling_createqueue.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengaktifkan Polling Panjang pada Antrian yang Ada

Buat modul Node.js dengan nama file`sqs_longpolling_existingqueue.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon Simple Queue Service, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk mengatur atribut antrian, termasuk nilai bukan nol untuk `ReceiveMessageWaitTimeSeconds` parameter dan URL antrian. Panggil metode `setQueueAttributes`. Polling panjang kemudian diaktifkan untuk antrian.

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

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_longpolling_existingqueue.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Mengaktifkan Polling Panjang pada Tanda Terima Pesan

Buat modul Node.js dengan nama file`sqs_longpolling_receivemessage.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon Simple Queue Service, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk menerima pesan, termasuk nilai bukan nol untuk `WaitTimeSeconds` parameter dan URL antrian. Panggil metode `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);
  }
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_longpolling_receivemessage.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Menggunakan Antrian Surat Mati di Amazon SQS

Contoh kode Node.js ini menunjukkan:

- Cara menggunakan antrian untuk menerima dan menahan pesan dari antrian lain yang tidak dapat diproses antrian.

Skenario

Antrian surat mati adalah antrian yang dapat ditargetkan oleh antrian (sumber) lain untuk pesan yang tidak dapat diproses dengan sukses. Anda dapat menyisihkan dan mengisolasi pesan-pesan ini dalam antrian surat mati untuk menentukan mengapa pemrosesan mereka tidak berhasil. Anda harus mengkonfigurasi secara individual setiap antrian sumber yang mengirim pesan ke antrian huruf mati. Beberapa antrian dapat menargetkan antrian huruf mati tunggal.

Dalam contoh ini, modul Node.js digunakan untuk merutekan pesan ke antrian huruf mati. Modul Node.js menggunakan SDK JavaScript untuk menggunakan antrian huruf mati menggunakan metode kelas klien ini: `AWS.SQS`

- `setQueueAttributes`

Untuk informasi selengkapnya tentang antrian surat mati Amazon SQS, lihat Menggunakan Antrian Surat Mati Amazon SQS di Panduan Pengembang Layanan Antrian Sederhana Amazon.

Tugas Prasyarat

Untuk mengatur dan menjalankan contoh ini, Anda harus terlebih dahulu menyelesaikan tugas-tugas ini:

- Instal Node.js. Untuk informasi selengkapnya tentang menginstal Node.js, lihat situs web Node.js.
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat. Memuat Kredensial di Node.js dari File Kredensial Bersama

- Buat antrian Amazon SQS untuk berfungsi sebagai antrian surat mati. Untuk contoh membuat antrian, lihatMenggunakan Antrian di Amazon SQS.

Mengkonfigurasi Antrian Sumber

Setelah Anda membuat antrian untuk bertindak sebagai antrian huruf mati, Anda harus mengonfigurasi antrian lain yang merutekan pesan yang belum diproses ke antrian huruf mati. Untuk melakukan ini, tentukan kebijakan redrive yang mengidentifikasi antrian yang akan digunakan sebagai antrian huruf mati dan jumlah maksimum yang diterima oleh masing-masing pesan sebelum dialihkan ke antrian huruf mati.

Buat modul Node.js dengan nama file`sqs_deadletterqueue.js`. Pastikan untuk mengkonfigurasi SDK seperti yang ditunjukkan sebelumnya. Untuk mengakses Amazon SQS, buat objek `AWS.SQS` layanan. Buat objek JSON yang berisi parameter yang diperlukan untuk memperbarui atribut antrian, termasuk `RedrivePolicy` parameter yang menentukan ARN dari antrian huruf mati, serta nilai. `maxReceiveCount` Juga tentukan antrian sumber URL yang ingin Anda konfigurasikan. Panggil metode `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);
  }
```

```
});
```

Untuk menjalankan contoh, ketik berikut ini di baris perintah.

```
node sqs_deadletterqueue.js
```

Kode contoh ini dapat ditemukan di sini GitHub.

Tutorial

Tutorial berikut menunjukkan kepada Anda bagaimana melakukan berbagai tugas yang terkait dengan penggunaan AWS SDK untuk JavaScript.

Topik

- Tutorial: Menyiapkan Node.js pada EC2 Instans Amazon

Tutorial: Menyiapkan Node.js pada EC2 Instans Amazon

Skenario umum untuk menggunakan Node.js dengan SDK for JavaScript adalah menyiapkan dan menjalankan aplikasi web Node.js pada instance Amazon Elastic Compute Cloud EC2 (Amazon). Dalam tutorial ini, Anda akan membuat instance Linux, menghubungkannya menggunakan SSH, dan kemudian menginstal Node.js untuk menjalankan instance itu.

Prasyarat

Tutorial ini mengasumsikan bahwa Anda telah meluncurkan instance Linux dengan nama DNS publik yang dapat dijangkau dari Internet dan yang dapat Anda sambungkan menggunakan SSH. Untuk informasi selengkapnya, lihat Langkah 1: Meluncurkan Instance di Panduan EC2 Pengguna Amazon.

> **Important**
>
> Gunakan Amazon Linux 2023 Amazon Machine Image (AMI) saat meluncurkan EC2 instans Amazon baru.

Anda juga harus mengonfigurasi grup keamanan Anda untuk mengizinkan koneksi SSH (port 22), HTTP (port 80), dan HTTPS (port 443). Untuk informasi selengkapnya tentang prasyarat ini, lihat Menyiapkan dengan Amazon Amazon di Panduan Pengguna EC2 Amazon. EC2

Prosedur

Prosedur berikut membantu Anda menginstal Node.js pada instance Amazon Linux. Anda dapat menggunakan server ini untuk meng-host aplikasi web Node.js.

Untuk mengatur Node.js pada instance Linux Anda

1. Connect ke instance Linux Anda seperti `ec2-user` menggunakan SSH.
2. Instal node version manager (nvm) dengan mengetikkan berikut ini di baris perintah.

   > **Warning**
   >
   > AWS tidak mengontrol kode berikut. Sebelum Anda menjalankannya, pastikan untuk memverifikasi keaslian dan integritasnya. Informasi lebih lanjut tentang kode ini dapat ditemukan di repositori nvm GitHub .

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
   ```

   Kami akan menggunakan nvm untuk menginstal Node.js karena nvm dapat menginstal beberapa versi Node.js dan memungkinkan Anda untuk beralih di antara mereka.
3. Muat `nvm` dengan mengetikkan berikut ini di baris perintah.

   ```
   source ~/.bashrc
   ```
4. Gunakan nvm untuk menginstal versi LTS terbaru dari Node.js dengan mengetikkan berikut ini di baris perintah.

   ```
   nvm install --lts
   ```

   Menginstal Node.js juga menginstal Node Package Manager (npm), sehingga Anda dapat menginstal modul tambahan sesuai kebutuhan.
5. Uji bahwa Node.js diinstal dan berjalan dengan benar dengan mengetikkan berikut ini di baris perintah.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Ini menampilkan pesan berikut yang menunjukkan versi Node.js yang sedang berjalan.

   ```
   Running Node.js VERSION
   ```

> **Note**
>
> Instalasi node hanya berlaku untuk EC2 sesi Amazon saat ini. Jika Anda memulai ulang sesi CLI Anda, Anda perlu menggunakan nvm untuk mengaktifkan versi node yang diinstal. Jika instance dihentikan, Anda perlu menginstal node lagi. Alternatifnya adalah membuat Amazon Machine Image (AMI) dari EC2 instance Amazon setelah Anda memiliki konfigurasi yang ingin Anda simpan, seperti yang dijelaskan dalam topik berikut.

Membuat Gambar Mesin Amazon

Setelah menginstal Node.js pada EC2 instance Amazon, Anda dapat membuat Amazon Machine Image (AMI) dari instance tersebut. Membuat AMI memudahkan penyediaan beberapa EC2 instans Amazon dengan instalasi Node.js yang sama. Untuk informasi selengkapnya tentang membuat AMI dari instans yang ada, lihat Membuat AMI Linux yang didukung Amazon EBS di EC2 Panduan Pengguna Amazon.

Sumber Daya Terkait

Untuk informasi selengkapnya tentang perintah dan perangkat lunak yang digunakan dalam topik ini, lihat halaman web berikut:

- manajer versi node (nvm): lihat repo nvm aktif. GitHub
- manajer paket node (npm): lihat situs web npm.

JavaScript Referensi API

Topik Referensi API untuk versi terbaru SDK untuk dapat JavaScript ditemukan di:

AWS SDK untuk JavaScript Panduan Referensi API.

SDK Changelog aktif GitHub

Changelog untuk rilis dari versi 2.4.8 dan yang lebih baru ditemukan di:

Ubah log.

Migrasi ke v3 dari AWS SDK untuk JavaScript

AWS SDK untuk JavaScript Versi 3 adalah penulisan ulang utama versi 2. Untuk informasi selengkapnya tentang migrasi ke versi 3, lihat Memigrasi dari versi 2.x ke 3.x AWS SDK untuk JavaScript di Panduan Pengembang v3.AWS SDK untuk JavaScript

Keamanan untuk AWS Produk atau Layanan ini

Keamanan cloud di Amazon Web Services (AWS) merupakan prioritas tertinggi. Sebagai seorang pelanggan AWS , Anda mendapatkan manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan dari organisasi yang paling sensitif terhadap keamanan. Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model Tanggung Jawab Bersama menggambarkan ini sebagai Keamanan dari Cloud dan Keamanan dalam Cloud.

Security of the Cloud - AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan semua layanan yang ditawarkan di AWS Cloud dan memberi Anda layanan yang dapat Anda gunakan dengan aman. Tanggung jawab keamanan kami adalah prioritas tertinggi di AWS, dan efektivitas keamanan kami secara teratur diuji dan diverifikasi oleh auditor pihak ketiga sebagai bagian dari Program AWS Kepatuhan.

Keamanan di Cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan, dan faktor-faktor lain termasuk sensitivitas data Anda, persyaratan organisasi Anda, serta undang-undang dan peraturan yang berlaku.

AWS Produk atau layanan ini mengikuti model tanggung jawab bersama melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat halaman dokumentasi keamanan AWS layanan dan AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan.

Topik

- Perlindungan data dalam AWS produk atau layanan ini
- Identity and Access Management
- Validasi Kepatuhan untuk AWS Produk atau Layanan ini
- Ketahanan untuk AWS Produk atau Layanan ini
- Keamanan Infrastruktur untuk AWS Produk atau Layanan ini
- Menegakkan versi minimum TLS

Perlindungan data dalam AWS produk atau layanan ini

Model tanggung jawab AWS bersama model berlaku untuk perlindungan data dalam AWS produk atau layanan ini. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk

melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Lihat informasi yang lebih lengkap tentang privasi data dalam Pertanyaan Umum Privasi Data. Lihat informasi tentang perlindungan data di Eropa di pos blog Model Tanggung Jawab Bersama dan GDPR AWS di Blog Keamanan AWS .

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensyal dan mengatur pengguna individu dengan AWS IAM Identity Center atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan sumber daya. AWS Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan CloudTrail jejak untuk menangkap AWS aktivitas, lihat Bekerja dengan CloudTrail jejak di AWS CloudTrail Panduan Pengguna.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-3 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-3.

Kami sangat merekomendasikan agar Anda tidak pernah memasukkan informasi identifikasi yang sensitif, seperti nomor rekening pelanggan Anda, ke dalam tanda atau bidang isian bebas seperti bidang Nama. Ini termasuk ketika Anda bekerja dengan AWS produk atau layanan ini atau lainnya Layanan AWS menggunakan konsol, API AWS CLI, atau AWS SDKs. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau log diagnostik. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Identity and Access Management

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diberi wewenang (memiliki izin) untuk menggunakan sumber daya. AWS IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- Audiens
- Mengautentikasi dengan identitas
- Mengelola akses menggunakan kebijakan
- Bagaimana Layanan AWS bekerja dengan IAM
- Memecahkan masalah AWS identitas dan akses

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan. AWS

Pengguna layanan — Jika Anda menggunakan Layanan AWS untuk melakukan pekerjaan Anda, maka administrator Anda memberi Anda kredensyal dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak AWS fitur untuk melakukan pekerjaan Anda, Anda mungkin memerlukan izin tambahan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur AWS, lihat Memecahkan masalah AWS identitas dan akses atau panduan pengguna yang Layanan AWS Anda gunakan.

Administrator layanan — Jika Anda bertanggung jawab atas AWS sumber daya di perusahaan Anda, Anda mungkin memiliki akses penuh ke AWS. Tugas Anda adalah menentukan AWS fitur dan sumber daya mana yang harus diakses pengguna layanan Anda. Kemudian, Anda harus mengirimkan permintaan kepada administrator IAM untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep dasar IAM. Untuk mempelajari lebih lanjut tentang bagaimana perusahaan Anda dapat menggunakan IAM AWS, lihat panduan pengguna yang Layanan AWS Anda gunakan.

Administrator IAM – Jika Anda adalah administrator IAM, Anda mungkin ingin belajar dengan lebih detail tentang cara Anda menulis kebijakan untuk mengelola akses ke AWS. Untuk melihat contoh

kebijakan AWS berbasis identitas yang dapat Anda gunakan di IAM, lihat panduan pengguna yang Anda gunakan. Layanan AWS

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensyal identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensyal yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (IAM Identity Center), autentikasi masuk tunggal perusahaan Anda, dan kredensyal Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas terfederasi, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan peran IAM. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS.

Jika Anda mengakses AWS secara terprogram, AWS sediakan kit pengembangan perangkat lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensil Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Guna mengetahui informasi selengkapnya tentang penggunaan metode yang disarankan untuk menandatangani permintaan sendiri, lihat AWS Signature Version 4 untuk permintaan API dalam Panduan Pengguna IAM.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari selengkapnya, lihat Autentikasi multi-faktor dalam Panduan Pengguna AWS IAM Identity Center dan Autentikasi multi-faktor AWS di IAM dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut

untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas yang mengharuskan Anda masuk sebagai pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas gabungan

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensyal sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensil yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensyal sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat Identitas IAM, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat Identitas IAM, lihat Apakah itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, kami merekomendasikan untuk mengandalkan kredensial sementara, bukan membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan tertentu yang memerlukan kredensial jangka panjang dengan pengguna IAM, kami merekomendasikan Anda merotasi kunci akses. Untuk informasi selengkapnya, lihat Merotasi kunci akses secara teratur untuk kasus penggunaan yang memerlukan kredensial jangka panjang dalam Panduan Pengguna IAM.

Grup IAM adalah identitas yang menentukan sekumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin bagi beberapa pengguna sekaligus. Grup mempermudah manajemen izin untuk sejumlah besar pengguna sekaligus. Misalnya, Anda dapat meminta kelompok untuk menyebutkan IAMAdmins dan memberikan izin kepada grup tersebut untuk mengelola sumber daya IAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna

memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari selengkapnya, lihat Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Peran ini mirip dengan pengguna IAM, tetapi tidak terkait dengan orang tertentu. Untuk mengambil peran IAM sementara AWS Management Console, Anda dapat beralih dari pengguna ke peran IAM (konsol). Anda dapat mengambil peran dengan memanggil operasi AWS CLI atau AWS API atau dengan menggunakan URL kustom. Untuk informasi selengkapnya tentang cara menggunakan peran, lihat Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna terfederasi – Untuk menetapkan izin ke identitas terfederasi, Anda membuat peran dan menentukan izin untuk peran tersebut. Ketika identitas terfederasi mengautentikasi, identitas tersebut terhubung dengan peran dan diberi izin yang ditentukan oleh peran. Untuk informasi tentang peran untuk federasi, lihat Buat peran untuk penyedia identitas pihak ketiga dalam Panduan Pengguna IAM. Jika menggunakan Pusat Identitas IAM, Anda harus mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM akan mengorelasikan set izin ke peran dalam IAM. Untuk informasi tentang set izin, lihat Set izin dalam Panduan Pengguna AWS IAM Identity Center .
- Izin pengguna IAM sementara – Pengguna atau peran IAM dapat mengambil peran IAM guna mendapatkan berbagai izin secara sementara untuk tugas tertentu.
- Akses lintas akun – Anda dapat menggunakan peran IAM untuk mengizinkan seseorang (prinsipal tepercaya) di akun lain untuk mengakses sumber daya di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa Layanan AWS, Anda dapat melampirkan kebijakan secara langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Misalnya, saat Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.

- Sesi akses teruskan (FAS) — Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat Sesi akses maju.
- Peran layanan – Peran layanan adalah peran IAM yang dijalankan oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.
- Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan peran IAM untuk mengelola kredensi sementara untuk aplikasi yang berjalan pada EC2 instance dan membuat AWS CLI atau AWS permintaan API. Ini lebih baik untuk menyimpan kunci akses dalam EC2 instance. Untuk menetapkan AWS peran ke EC2 instance dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instance berisi peran dan memungkinkan program yang berjalan pada EC2 instance untuk mendapatkan kredensyal sementara. Untuk informasi selengkapnya, lihat Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di EC2 instans Amazon di Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika prinsipal (pengguna, pengguna root, atau sesi peran) membuat permintaan. Izin dalam kebijakan menentukan apakah permintaan diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang struktur dan isi dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat mengambil peran.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk melakukan operasinya. Misalnya, anggaplah Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan tersebut bisa mendapatkan informasi peran dari AWS Management Console, API AWS CLI, atau AWS API.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan yang dikelola. Kebijakan inline disematkan langsung ke satu pengguna, grup, atau peran. Kebijakan terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa pengguna, grup, dan peran dalam. Akun AWS Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan yang dikelola atau kebijakan inline, lihat Pilih antara kebijakan yang dikelola dan kebijakan inline dalam Panduan Pengguna IAM.

Kebijakan berbasis sumber daya

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus

menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACLs)

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung. ACLs Untuk mempelajari selengkapnya ACLs, lihat Ringkasan daftar kontrol akses (ACL) di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Jenis-jenis kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda oleh jenis kebijakan yang lebih umum.

- Batasan izin – Batasan izin adalah fitur lanjutan tempat Anda mengatur izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas ke entitas IAM (pengguna IAM atau peran IAM). Anda dapat menetapkan batasan izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izinnya. Kebijakan berbasis sumber daya yang menentukan pengguna atau peran dalam bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian izin. Untuk informasi selengkapnya tentang batasan izin, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCPs) — SCPs adalah kebijakan JSON yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di. AWS Organizations AWS Organizations adalah layanan untuk mengelompokkan dan mengelola secara terpusat beberapa Akun AWS yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur dalam suatu organisasi, maka Anda dapat menerapkan kebijakan kontrol layanan (SCPs) ke salah satu atau semua akun Anda. SCP membatasi izin untuk entitas di akun anggota, termasuk masing-masing. Pengguna root akun AWS Untuk informasi selengkapnya tentang Organizations dan SCPs, lihat Kebijakan kontrol layanan di Panduan AWS Organizations Pengguna.
- Kebijakan kontrol sumber daya (RCPs) — RCPs adalah kebijakan JSON yang dapat Anda gunakan untuk menetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda tanpa

memperbarui kebijakan IAM yang dilampirkan ke setiap sumber daya yang Anda miliki. RCP membatasi izin untuk sumber daya di akun anggota dan dapat memengaruhi izin efektif untuk identitas, termasuk Pengguna root akun AWS, terlepas dari apakah itu milik organisasi Anda. Untuk informasi selengkapnya tentang Organizations dan RCPs, termasuk daftar dukungan Layanan AWS tersebut RCPs, lihat Kebijakan kontrol sumber daya (RCPs) di Panduan AWS Organizations Pengguna.

- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda berikan sebagai parameter ketika Anda membuat sesi sementara secara programatis untuk peran atau pengguna terfederasi. Izin sesi yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas pengguna atau peran dan kebijakan sesi. Izin juga bisa datang dari kebijakan berbasis sumber daya. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian izin. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Bagaimana Layanan AWS bekerja dengan IAM

Untuk mendapatkan tampilan tingkat tinggi tentang cara Layanan AWS bekerja dengan sebagian besar fitur IAM, lihat AWS layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Untuk mempelajari cara menggunakan yang spesifik Layanan AWS dengan IAM, lihat bagian keamanan dari Panduan Pengguna layanan yang relevan.

Memecahkan masalah AWS identitas dan akses

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan AWS dan IAM.

Topik

- Saya tidak berwenang untuk melakukan tindakan di AWS
- Saya tidak berwenang untuk melakukan iam: PassRole
- Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses AWS sumber daya saya

Saya tidak berwenang untuk melakukan tindakan di AWS

Jika Anda menerima pesan kesalahan bahwa Anda tidak memiliki otorisasi untuk melakukan tindakan, kebijakan Anda harus diperbarui agar Anda dapat melakukan tindakan tersebut.

Contoh kesalahan berikut terjadi ketika pengguna IAM `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya *`my-example-widget`* rekaan, tetapi tidak memiliki izin `awes:`*`GetWidget`* rekaan.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Dalam hal ini, kebijakan untuk pengguna `mateojackson` harus diperbarui untuk mengizinkan akses ke sumber daya *`my-example-widget`* dengan menggunakan tindakan `awes:`*`GetWidget`*.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan yang tidak diizinkan untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran AWS.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol tersebut untuk melakukan tindakan di AWS. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses AWS sumber daya saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mempelajari apakah AWS mendukung fitur-fitur ini, lihatBagaimana Layanan AWS bekerja dengan IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

Validasi Kepatuhan untuk AWS Produk atau Layanan ini

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat Layanan AWS di Lingkup oleh Program Kepatuhan Layanan AWS dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat Program AWS Kepatuhan Program AWS .

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh Laporan di AWS Artifact .

Tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- Kepatuhan dan Tata Kelola Keamanan – Panduan implementasi solusi ini membahas pertimbangan arsitektur serta memberikan langkah-langkah untuk menerapkan fitur keamanan dan kepatuhan.
- Referensi Layanan yang Memenuhi Syarat HIPAA — Daftar layanan yang memenuhi syarat HIPAA. Tidak semua memenuhi Layanan AWS syarat HIPAA.
- AWS Sumber Daya AWS — Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- AWS Panduan Kepatuhan Pelanggan - Memahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan untuk kontrol keamanan di berbagai kerangka kerja (termasuk Institut Standar dan Teknologi Nasional (NIST), Dewan Standar Keamanan Industri Kartu Pembayaran (PCI), dan Organisasi Internasional untuk Standardisasi (ISO)).
- Mengevaluasi Sumber Daya dengan Aturan dalam Panduan AWS Config Pengembang — AWS Config Layanan menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub— Ini Layanan AWS memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS. Security Hub menggunakan kontrol keamanan untuk sumber daya AWS Anda serta untuk memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat Referensi kontrol Security Hub.
- Amazon GuardDuty — Ini Layanan AWS mendeteksi potensi ancaman terhadap beban kerja Akun AWS, kontainer, dan data Anda dengan memantau lingkungan Anda untuk aktivitas mencurigakan dan berbahaya. GuardDuty dapat membantu Anda mengatasi berbagai persyaratan kepatuhan, seperti PCI DSS, dengan memenuhi persyaratan deteksi intrusi yang diamanatkan oleh kerangka kerja kepatuhan tertentu.
- AWS Audit ManagerIni Layanan AWS membantu Anda terus mengaudit AWS penggunaan Anda untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap peraturan dan standar industri.

AWS Produk atau layanan ini mengikuti model tanggung jawab bersama melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat halaman dokumentasi keamanan AWS layanan dan AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan.

Ketahanan untuk AWS Produk atau Layanan ini

Infrastruktur AWS global dibangun di sekitar AWS Region dan Availability Zones.

AWS Region menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan.

Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara zona tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat Infrastruktur AWS Global.

AWS Produk atau layanan ini mengikuti model tanggung jawab bersama melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat halaman dokumentasi keamanan AWS layanan dan AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan.

Keamanan Infrastruktur untuk AWS Produk atau Layanan ini

AWS Produk atau layanan ini menggunakan layanan terkelola, dan karenanya dilindungi oleh keamanan jaringan AWS global. Untuk informasi tentang layanan AWS keamanan dan cara AWS melindungi infrastruktur, lihat Keamanan AWS Cloud. Untuk mendesain AWS lingkungan Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat Perlindungan Infrastruktur dalam Kerangka Kerja yang AWS Diarsiteksikan dengan Baik Pilar Keamanan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses AWS Produk atau Layanan ini melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Sandi cocok dengan sistem kerahasiaan maju sempurna (perfect forward secrecy, PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan prinsipal IAM. Atau Anda dapat menggunakan AWS Security Token

Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk menandatangani permintaan.

AWS Produk atau layanan ini mengikuti model tanggung jawab bersama melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat halaman dokumentasi keamanan AWS layanan dan AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan.

Menegakkan versi minimum TLS

Untuk menambahkan peningkatan keamanan saat berkomunikasi dengan AWS layanan, konfigurasikan AWS SDK untuk JavaScript untuk menggunakan TLS 1.2 atau yang lebih baru.

Transport Layer Security (TLS) adalah protokol yang digunakan oleh browser web dan aplikasi lain untuk memastikan privasi dan integritas data yang dipertukarkan melalui jaringan.

> **Important**
>
> Mulai 10 Juni 2024, kami mengumumkan bahwa TLS 1.3 tersedia di titik akhir API AWS layanan di setiap Wilayah. AWS AWS SDK untuk JavaScript V2 tidak menegosiasikan versi TLS itu sendiri. Sebaliknya, ia menggunakan versi TLS yang ditentukan oleh Node.js, yang dapat dikonfigurasi melalui. `https.Agent` AWS merekomendasikan menggunakan versi Active LTS Node.js saat ini.

Verifikasi dan terapkan TLS di Node.js

Saat Anda menggunakan AWS SDK untuk JavaScript with Node.js, layer keamanan Node.js yang mendasarinya digunakan untuk mengatur versi TLS.

Node.js 12.0.0 dan yang lebih baru menggunakan versi minimum OpenSSL 1.1.1b, yang mendukung TLS 1.3. Default AWS SDK untuk JavaScript v2 menggunakan TLS 1.3 bila tersedia, tetapi default ke versi yang lebih rendah jika diperlukan.

Verifikasi versi OpenSSL dan TLS

Untuk mendapatkan versi OpenSSL yang digunakan oleh Node.js di komputer Anda, jalankan perintah berikut.

```
node -p process.versions
```

Versi OpenSSL dalam daftar adalah versi yang digunakan oleh Node.js, seperti yang ditunjukkan pada contoh berikut.

```
openssl: '1.1.1b'
```

Untuk mendapatkan versi TLS yang digunakan oleh Node.js di komputer Anda, jalankan shell Node dan jalankan perintah berikut, secara berurutan.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

Perintah terakhir menampilkan versi TLS, seperti yang ditunjukkan pada contoh berikut.

```
'TLSv1.3'
```

Node.js default untuk menggunakan versi TLS ini, dan mencoba menegosiasikan versi TLS lain jika panggilan tidak berhasil.

Memeriksa Versi TLS Minimum dan Maksimum yang Didukung

Pengembang dapat memeriksa versi TLS minimum dan maksimum yang didukung di Node.js menggunakan skrip berikut:

```
var tls = require("tls");
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +
 tls.DEFAULT_MAX_VERSION);
```

Perintah terakhir menampilkan versi TLS minimum dan maksimum default, seperti yang ditunjukkan pada contoh berikut.

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

Menerapkan versi minimum TLS

Node.js menegosiasikan versi TLS saat panggilan gagal. Anda dapat menerapkan versi TLS minimum yang diizinkan selama negosiasi ini, baik saat menjalankan skrip dari baris perintah atau per permintaan dalam kode Anda. JavaScript

Untuk menentukan versi TLS minimum dari baris perintah, Anda harus menggunakan Node.js versi 11.4.0 atau yang lebih baru. Untuk menginstal versi Node.js tertentu, pertama instal Node Version Manager (nvm) menggunakan langkah-langkah yang ditemukan di Node Version Manager Installating and Update. Kemudian jalankan perintah berikut untuk menginstal dan menggunakan versi tertentu dari Node.js.

```
nvm install 11
nvm use 11
```

Enforcing TLS 1.2

Untuk menegakkan bahwa TLS 1.2 adalah versi minimum yang diizinkan, tentukan `--tls-min-v1.2` argumen saat menjalankan skrip Anda, seperti yang ditunjukkan pada contoh berikut.

```
node --tls-min-v1.2 yourScript.js
```

Untuk menentukan versi TLS minimum yang diizinkan untuk permintaan tertentu dalam JavaScript kode Anda, gunakan `httpOptions` parameter untuk menentukan protokol, seperti yang ditunjukkan pada contoh berikut.

```
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

Untuk menegakkan bahwa TLS 1.3 adalah versi minimum yang diizinkan, tentukan `--tls-min-v1.3` argumen saat menjalankan skrip Anda, seperti yang ditunjukkan pada contoh berikut.

```
node --tls-min-v1.3 yourScript.js
```

Untuk menentukan versi TLS minimum yang diizinkan untuk permintaan tertentu dalam JavaScript kode Anda, gunakan `httpOptions` parameter untuk menentukan protokol, seperti yang ditunjukkan pada contoh berikut.

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

Verifikasi dan terapkan TLS dalam skrip browser

Saat Anda menggunakan SDK for JavaScript dalam skrip browser, pengaturan browser mengontrol versi TLS yang digunakan. Versi TLS yang digunakan oleh browser tidak dapat ditemukan atau diatur oleh skrip dan harus dikonfigurasi oleh pengguna. Untuk memverifikasi dan menerapkan versi TLS yang digunakan dalam skrip browser, lihat instruksi untuk browser spesifik Anda.

Microsoft Internet Explorer

1. Buka Internet Explorer.
2. Dari bilah menu, pilih Tools - Internet Options - Advanced tab.
3. Gulir ke bawah ke kategori Keamanan, centang kotak opsi secara manual untuk Gunakan TLS 1.2.

4. Klik OK.
5. Tutup browser Anda dan mulai ulang Internet Explorer.

Microsoft Edge

1. Di kotak pencarian menu Windows, ketik*Internet options*.
2. Di bawah Best match, klik Internet Options.
3. Di jendela Internet Properties, pada tab Advanced, gulir ke bawah ke bagian Keamanan.
4. Centang kotak centang User TLS 1.2.
5. Klik OK.

Google Chrome

1. Buka Google Chrome.
2. Klik Alt F dan pilih Pengaturan.
3. Gulir ke bawah dan pilih Tampilkan pengaturan lanjutan... .
4. Gulir ke bawah ke bagian Sistem dan klik Buka pengaturan proxy... .
5. Pilih tab Advanced.
6. Gulir ke bawah ke kategori Keamanan, centang kotak opsi secara manual untuk Gunakan TLS 1.2.
7. Klik OK.
8. Tutup browser Anda dan mulai ulang Google Chrome.

Mozilla Firefox

1. Buka Firefox.
2. Di bilah alamat, ketik about:config dan tekan Enter.
3. Di bidang Pencarian, masukkan tls. Temukan dan klik dua kali entri untuk security.tls.version.min.
4. Atur nilai integer ke 3 untuk memaksa protokol TLS 1.2 menjadi default.
5. Klik OK.
6. Tutup browser Anda dan mulai ulang Mozilla Firefox.

Apple Safari

Tidak ada opsi untuk mengaktifkan protokol SSL. Jika Anda menggunakan Safari versi 7 atau lebih tinggi, TLS 1.2 diaktifkan secara otomatis.

Sumber Daya Tambahan

Tautan berikut menyediakan sumber daya tambahan yang dapat Anda gunakan dengan AWS SDK untuk JavaScript.

AWS SDKs dan Panduan Referensi Alat

Panduan Referensi AWS SDKs and Tools juga berisi pengaturan, fitur, dan konsep dasar lainnya yang umum di antara banyak. AWS SDKs

JavaScript Forum SDK

Anda dapat menemukan pertanyaan dan diskusi tentang hal-hal yang menarik bagi pengguna SDK untuk JavaScript di Forum JavaScript SDK.

JavaScript SDK dan Panduan Pengembang di GitHub

Ada beberapa repositori untuk SDK GitHub untuk. JavaScript

- SDK saat ini JavaScript tersedia di repo SDK.
- SDK for JavaScript Developer Guide (dokumen ini) tersedia dalam format penurunan harga dalam repo dokumentasinya sendiri.
- Beberapa kode contoh yang disertakan dalam panduan ini tersedia di repo kode sampel SDK.

JavaScript SDK di Gitter

Anda juga dapat menemukan pertanyaan dan diskusi tentang SDK untuk JavaScript di komunitas JavaScript SDK di Gitter.

Riwayat Dokumen untuk AWS SDK untuk JavaScript

- Versi SDK: Lihat JavaScript Referensi API
- Pembaruan dokumentasi utama terbaru: 31 Maret 2022

Riwayat Dokumen

Tabel berikut menjelaskan perubahan penting dalam setiap rilis AWS SDK untuk JavaScript setelah Mei 2018. Untuk notifikasi tentang pembaruan dokumentasi ini, Anda dapat berlangganan ke umpan RSS.

Perubahan	Deskripsi	Tanggal
TLS 1.3 sekarang didukung di semua titik akhir API AWS layanan di semua Wilayah	Diperbarui versi dan metode TLS yang didukung untuk mencatat versi TLS.	April 10, 2025
Menegakkan versi minimum TLS	Menambahkan informasi tentang TLS 1.3.	31 Maret 2022
Melihat Foto di Bucket Amazon S3 dari Browser	Menambahkan contoh untuk hanya melihat foto di album foto yang ada.	13 Mei 2019
Menyetel Kredensyal di Node.js, pilihan pemuatan kredensyal baru	Menambahkan informasi tentang kredenal yang dimuat dari penyedia kredensi ECS atau proses kredensi yang dikonfigurasi.	25 April 2019
Kredensyal menggunakan Proses Kredensyal yang Dikonfigurasi	Menambahkan informasi tentang kredensyal yang dimuat dari proses kredensyal yang dikonfigurasi.	25 April 2019
Baru Memulai di Skrip Browser	Memulai dalam Skrip Browser telah ditulis ulang untuk	Juli 14, 2018

	menyederhanakan contoh dan untuk mengakses layanan Amazon Polly untuk mengirim teks dan mengembalikan ucapan yang disintesis yang dapat Anda mainkan di browser. Lihat Memulai di Skrip Browser untuk konten baru.	
Sampel Kode Amazon SNS Baru	Empat contoh kode Node.js baru untuk bekerja dengan Amazon SNS telah ditambahk an. Lihat Contoh Amazon SNS untuk kode sampel.	29 Juni 2018
Baru Memulai di Node.js	Memulai di Node.js telah ditulis ulang untuk menggunak an kode sampel yang diperbarui dan untuk memberikan detail yang lebih besar dalam cara membuat `package.json` file serta kode Node.js itu sendiri. Lihat Memulai di Node.js untuk konten baru.	4 Juni 2018

Pembaruan Sebelumnya

Tabel berikut menjelaskan perubahan penting dalam setiap rilis AWS SDK untuk JavaScript sebelum Juni 2018.

Perubahan	Deskripsi	Tanggal
Sampel AWS Elemental MediaConvert kode baru	Tiga contoh kode Node.js baru untuk bekerja dengan AWS Elemental MediaConv	21 Mei 2018

Perubahan	Deskripsi	Tanggal
	ert telah ditambahkan. Lihat AWS Elemental MediaConvert Contoh kode sampel.	
Edit Baru pada GitHub Tombol	Header setiap topik sekarang menyediakan tombol yang membawa Anda ke versi penurunan harga dari topik yang sama GitHub sehingga Anda dapat memberikan pengeditan untuk meningkatkan akurasi dan kelengkapan panduan.	21 Februari 2018
Topik Baru pada Titik Akhir Kustom	Informasi telah ditambahkan pada format dan penggunaan titik akhir khusus untuk melakukan panggilan API. Lihat Menentukan Titik Akhir Kustom.	20 Februari 2018
SDK untuk Panduan JavaScript Pengembang di GitHub	SDK for JavaScript Developer Guide tersedia dalam format penurunan harga dalam repo dokumentasinya sendiri. Anda dapat memposting masalah yang Anda ingin panduan untuk mengatasi atau mengirimkan permintaan tarik untuk mengirimkan perubahan yang diusulkan.	16 Februari 2018

Perubahan	Deskripsi	Tanggal
Contoh kode Amazon DynamoDB baru	Contoh kode Node.js baru untuk memperbarui tabel DynamoDB menggunak an Document Client telah ditambahkan. Lihat Menggunakan Klien Dokumen DynamoDB kode sampel.	14 Februari 2018
Topik Baru tentang SDK Logging	Topik yang menjelaskan cara mencatat panggilan API yang dibuat dengan SDK for JavaScript telah ditambahkan, termasuk informasi tentang penggunaan logger pihak ketiga. Lihat AWS SDK untuk JavaScript Panggilan Pencatatan.	5 Februari 2018
Topik yang Diperbarui tentang Pengaturan Wilayah	Topik yang menjelaskan cara menyetel Wilayah yang digunakan dengan SDK telah diperbarui dan diperluas, termasuk informasi tentang urutan prioritas untuk menyetel Wilayah. Lihat Mengatur AWS Wilayah.	12 Desember 2017
Contoh Kode Amazon SES Baru	Bagian dengan contoh kode SDK telah diperbarui untuk menyertakan lima contoh baru untuk bekerja dengan Amazon SES. Untuk informasi selengkapnya tentang contoh kode ini, lihatContoh Layanan Email Sederhana Amazon.	9 November 2017

Perubahan	Deskripsi	Tanggal
Peningkatan Kegunaan	Berdasarkan pengujian kegunaan baru-baru ini, sejumlah perubahan telah dilakukan untuk meningkatkan kegunaan dokumentasi. • Sampel kode lebih jelas diidentifikasi sebagai target baik untuk browser atau eksekusi Node.js. • Tautan TOC tidak lagi langsung melompat ke konten web lain, termasuk Referensi API. • Termasuk lebih banyak penautan di bagian Memulai ke detail tentang mendapatk an AWS kredensyal. • Memberikan informasi selengkapnya tentang fitur Node.js umum yang diperlukan untuk menggunakan SDK. Untuk informasi selengkapnya, lihat Pertimbangan Node.js.	Agustus 9, 2017

Perubahan	Deskripsi	Tanggal
Contoh Kode DynamoDB Baru	Bagian dengan contoh kode SDK telah diperbarui untuk menulis ulang dua contoh sebelumnya serta menambahkan tiga contoh baru untuk bekerja dengan DynamoDB. Untuk informasi selengkapnya tentang contoh kode ini, lihatContoh Amazon DynamoDB.	21 Juni 2017
Contoh Kode IAM Baru	Bagian dengan contoh kode SDK telah diperbarui untuk menyertakan lima contoh baru untuk bekerja dengan IAM. Untuk informasi selengkapnya tentang contoh kode ini, lihatAWS Contoh IAM.	Desember 23, 2016
Contoh Kode SQS Baru CloudWatch dan Amazon	Bagian dengan contoh kode SDK telah diperbarui untuk menyertakan contoh baru untuk bekerja dengan CloudWatch dan dengan Amazon SQS. Untuk informasi lebih lanjut tentang contoh kode ini, lihat CloudWatch Contoh Amazon danAmazon SQS Contoh.	20 Desember 2016

Perubahan	Deskripsi	Tanggal
Contoh EC2 Kode Amazon Baru	Bagian dengan contoh kode SDK telah diperbarui untuk menyertakan lima contoh baru untuk bekerja dengan Amazon EC2. Untuk informasi selengkapnya tentang contoh kode ini, lihat EC2 Contoh Amazon.	15 Desember 2016
Daftar browser yang didukung dibuat lebih terlihat	Daftar browser yang didukung oleh SDK for JavaScript, yang sebelumnya ditemukan dalam topik Prasyarat, telah diberikan topiknya sendiri untuk membuatnya lebih terlihat dalam daftar isi.	16 Nopember 2016
Publikasi awal Panduan Pengembang baru	Panduan Pengembang sebelumnya sekarang tidak digunakan lagi. Panduan Pengembang baru telah ditata ulang untuk membuat informasi lebih mudah ditemukan. Ketika JavaScript skenario Node.js atau browser menyajikan pertimbangan khusus, itu diidentifikasi sebagaimana mestinya. Panduan ini juga memberika n contoh kode tambahan yang lebih terorganisir untuk membuatnya lebih mudah dan lebih cepat ditemukan.	Oktober 28, 2016