



Panduan Pengembang untuk SDK Versi 3

AWS SDK for JavaScript



AWS SDK for JavaScript: Panduan Pengembang untuk SDK Versi 3

Copyright © 2024 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 para pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan kekayaan masing-masing pemiliknya, yang mungkin berafiliasi, terkait dengan, atau disponsori oleh Amazon, atau mungkin tidak.

Table of Contents

.....	viii
Apa itu AWS SDK for JavaScript?	1
Pemeliharaan dan dukungan untuk versi utama SDK	1
Apa yang baru di Versi 3	2
Paket termodulasi	2
Tumpukan middleware baru	7
Menggunakan SDK dengan Node.js	8
Menggunakan SDK dengan AWS Cloud9	8
Menggunakan SDK dengan AWS Amplify	8
Menggunakan SDK dengan browser web	8
Menggunakan browser di V3	9
Kasus penggunaan umum	9
Tentang contoh	10
Sumber daya	10
Mulai	11
Otentikasi SDK dengan AWS	11
Memulai sesi portal AWS akses	12
Informasi otentikasi lebih lanjut	13
Memulai dengan Node.js	14
Skenario	14
Prasyarat	14
Langkah 1: Siapkan struktur paket dan instal paket klien	14
Langkah 2: Tambahkan impor dan kode SDK yang diperlukan	15
Langkah 3: Jalankan contoh	18
Memulai di browser	18
Skenario	19
Langkah 1: Buat kumpulan identitas Amazon Cognito dan peran IAM	19
Langkah 2: Tambahkan kebijakan ke peran IAM yang dibuat	20
Langkah 3: Tambahkan ember dan objek Amazon S3	21
Langkah 4: Siapkan kode browser	22
Langkah 5: Jalankan Contoh	23
Pembersihan	23
Siapkan SDK untuk JavaScript	24
Prasyarat	24

Menyiapkan lingkungan AWS Node.js	24
Browser web yang didukung	25
Instal SDK	26
Muat SDK	27
Konfigurasi SDK untuk JavaScript	28
Konfigurasi per layanan	28
Tetapkan konfigurasi per layanan	29
Mengatur AWS Wilayah	29
Dalam konstruktor kelas klien	30
Gunakan variabel lingkungan	30
Gunakan file konfigurasi bersama	30
Urutan prioritas untuk pengaturan Wilayah	31
Tetapkan kredensialnya	31
Praktik terbaik untuk kredensial	31
Setel kredensial di Node.js	32
Mengatur kredensial di browser web	35
Pertimbangan Node.js	39
Gunakan modul Node.js bawaan	39
Gunakan paket npm	40
Konfigurasi MaxSockets di Node.js	40
Gunakan kembali koneksi dengan keep-alive di Node.js	41
Konfigurasi proxy untuk Node.js	42
Daftarkan bundel sertifikat di Node.js	42
Pertimbangan Skrip Browser	43
Membangun SDK untuk Browser	43
Cross-origin resource sharing (CORS)	44
Bundel dengan webpack	48
Bekerja dengan AWS layanan	53
Membuat dan memanggil objek layanan	53
Tentukan parameter objek layanan	54
Layanan panggilan secara asinkron	54
Kelola panggilan asinkron	55
Gunakan async/await	56
Gunakan janji	57
Menggunakan fungsi callback	58
Buat permintaan klien layanan	59

Menangani tanggapan klien layanan	60
Akses data yang dikembalikan dalam respons	60
Mengakses informasi kesalahan	61
Bekerja dengan JSON	61
JSON sebagai parameter objek layanan	62
Contoh kode subset dengan panduan	63
JavaScript ES6/CommonJS sintaks	64
Contoh Amazon DynamoDB	67
Contoh AWS Elemental MediaConvert	91
Contoh AWS Lambda	113
Contoh Amazon Lex	114
Contoh Amazon Polly	114
Contoh Amazon Redshift	118
Contoh Amazon SES	126
Contoh Amazon SNS	154
Contoh Amazon Transcribe	188
Cross-service: Menyiapkan Node.js pada instans Amazon EC2	200
Cross-service: Aplikasi untuk mengirimkan data	202
Cross-service: Aplikasi transkripsi	210
Lintas layanan: Amazon API Gateway dan Lambda	222
Cross-service: Alur kerja tanpa server dengan Step Functions	237
Cross-service: Acara Lambda terjadwal	253
Lintas layanan: Contoh Amazon Lex	265
Cross-service: Aplikasi pemesanan	278
Gunakan AWS Cloud9 dengan SDK untuk JavaScript	292
Langkah 1: Siapkan AWS akun Anda untuk digunakan AWS Cloud9	292
Langkah 2: Siapkan lingkungan AWS Cloud9 pengembangan Anda	293
Langkah 3: Siapkan SDK untuk JavaScript	293
Untuk menyiapkan SDK JavaScript untuk Node.js	293
Untuk mengatur SDK untuk JavaScript di browser	294
Langkah 4: Unduh kode contoh	294
Langkah 5: Jalankan dan debug kode contoh	295
Contoh kode	296
Tindakan dan skenario	296
Auto Scaling	298
Amazon Bedrock	341

Runtime Amazon Bedrock	345
Agen untuk Amazon Bedrock	360
Agen untuk Amazon Bedrock Runtime	374
CloudWatch	377
CloudWatch Acara	393
CloudWatch Log	400
CodeBuild	417
Penyedia Identitas Amazon Cognito	421
DynamoDB	440
Amazon EC2	487
Penyeimbang Beban Elastis	569
EventBridge	618
AWS Glue	625
HealthImaging	649
IAM	686
Lambda	795
Amazon Personalize	805
Kejadian Amazon Personalize	822
Waktu Aktif Amazon Personalize	826
Amazon Pinpoint	830
Amazon Redshift	840
Amazon S3	845
S3 Glacier	884
SageMaker	888
Secrets Manager	921
Amazon SES	923
Amazon SNS	945
Amazon SQS	985
Step Functions	1021
AWS STS	1023
AWS Support	1026
Amazon Transcribe	1044
Contoh lintas layanan	1052
Membangun aplikasi Amazon Transcribe	1053
Membangun aplikasi streaming Amazon Transcribe	1054
Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB	1054

Membangun chatbot Amazon Lex	1055
Membuat aplikasi nirserver untuk mengelola foto	1055
Membuat aplikasi web untuk melacak data DynamoDB	1056
Buat pelacak butir kerja Aurora Serverless	1056
Membuat aplikasi penjelajah Amazon Textract	1057
Buat aplikasi untuk menganalisis umpan balik pelanggan	1057
Mendeteksi APD dalam gambar	1061
Mendeteksi objek dalam gambar	1062
Mendeteksi orang dan objek dalam video	1063
Menginvokasi fungsi Lambda dari browser	1064
Menggunakan API Gateway untuk menginvokasi fungsi Lambda	1064
Menggunakan Step Functions untuk menginvokasi fungsi Lambda	1065
Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda	1066
Keamanan	1067
Perlindungan data	1067
Manajemen Identitas dan Akses	1069
Audiens	1069
Mengautentikasi dengan identitas	1070
Mengelola akses menggunakan kebijakan	1073
Bagaimana Layanan AWS bekerja dengan IAM	1076
Memecahkan masalah AWS identitas dan akses	1076
Validasi Kepatuhan	1078
Ketangguhan	1079
Keamanan Infrastruktur	1080
Menerapkan versi TLS minimum	1081
Verifikasi dan terapkan TLS di Node.js	1081
Verifikasi dan terapkan TLS dalam skrip browser	1084
Migrasi ke versi 3	1086
Migrasi ke V3	1086
Gunakan codemod untuk memigrasikan kode v2 yang ada	1086
Riwayat dokumen	1088
Riwayat Dokumen	1088

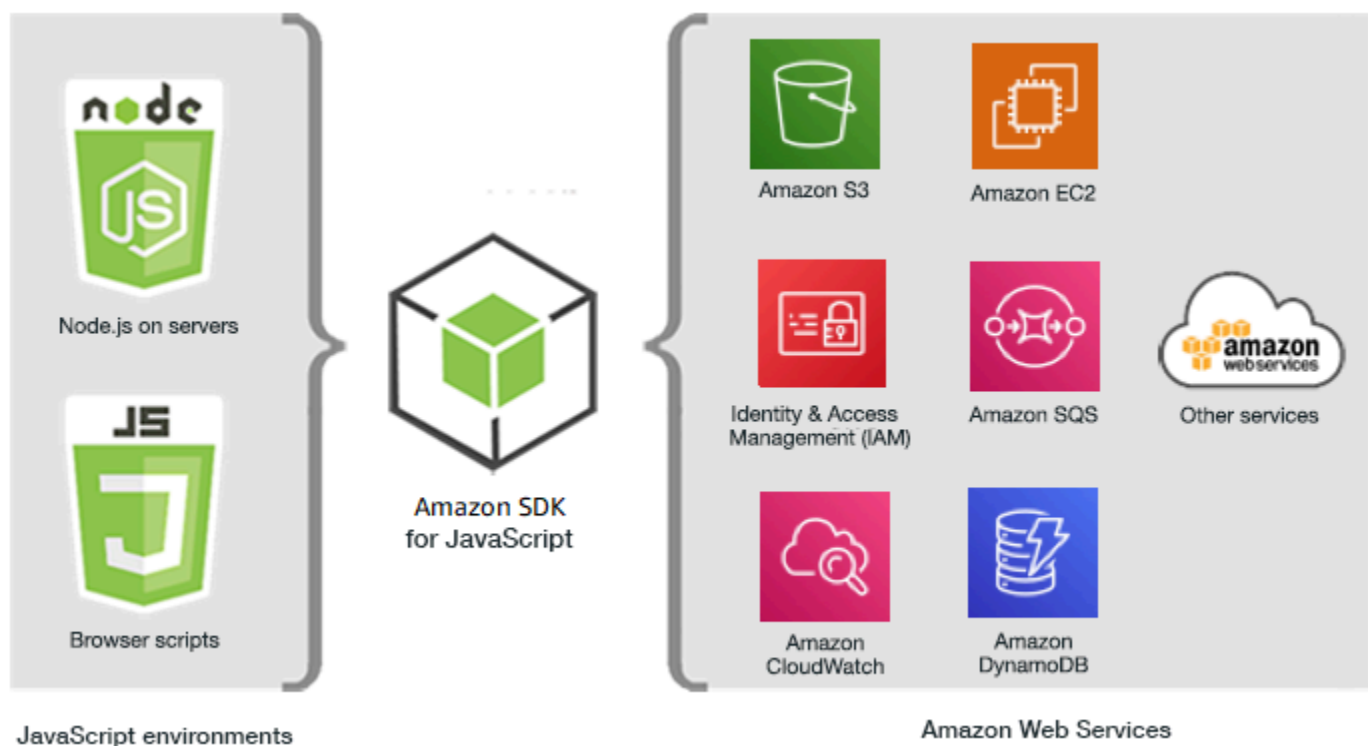
[Panduan Referensi API AWS SDK for JavaScript V3](#) menjelaskan secara rinci semua operasi API untuk AWS SDK for JavaScript versi 3 (V3).

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

Apa itu AWS SDK for JavaScript?

Selamat datang di Panduan AWS SDK for JavaScript Pengembang. Panduan ini memberikan informasi umum tentang pengaturan dan konfigurasi. AWS SDK for JavaScript Ini juga memandu Anda melalui contoh dan tutorial menjalankan berbagai AWS layanan menggunakan AWS SDK for JavaScript.

[Panduan Referensi API AWS SDK for JavaScript v3](#) menyediakan JavaScript API untuk AWS layanan. Anda dapat menggunakan JavaScript API untuk membangun pustaka atau aplikasi untuk [Node.js](#) atau browser.



Pemeliharaan dan dukungan untuk versi utama SDK

Untuk informasi tentang pemeliharaan dan dukungan untuk versi utama SDK dan dependensi yang mendasarinya, lihat berikut di [Panduan Referensi SDK dan Alat AWS](#):

- [AWS Kebijakan pemeliharaan SDK dan alat](#)
- [AWS Matriks dukungan versi SDK dan alat](#)

Apa yang baru di Versi 3

Versi 3 SDK for JavaScript (V3) berisi fitur-fitur baru berikut.

Paket termodulasi

Pengguna sekarang dapat menggunakan paket terpisah untuk setiap layanan.

Tumpukan middleware baru

Pengguna sekarang dapat menggunakan tumpukan middleware untuk mengontrol siklus hidup panggilan operasi.

Selain itu, SDK ditulis TypeScript, yang memiliki banyak keunggulan, seperti penyetoran statis.

Important

Contoh kode untuk V3 dalam panduan ini ditulis dalam ECMAScript 6 (ES6). ES6 menghadirkan sintaks baru dan fitur baru untuk membuat kode Anda lebih modern dan mudah dibaca, dan melakukan lebih banyak lagi. ES6 mengharuskan Anda menggunakan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduhannya Node.js](#). Untuk informasi selengkapnya, lihat [JavaScript ES6/CommonJS sintaks](#).

Paket termodulasi

Versi 2 SDK for JavaScript (V2) mengharuskan Anda untuk menggunakan seluruh AWS SDK, sebagai berikut.

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

Memuat seluruh SDK tidak menjadi masalah jika aplikasi Anda menggunakan banyak AWS layanan. Namun, jika Anda hanya perlu menggunakan beberapa AWS layanan, itu berarti meningkatkan ukuran aplikasi Anda dengan kode yang tidak Anda perlukan atau gunakan.

Di V3, Anda hanya dapat memuat dan menggunakan AWS Layanan individual yang Anda butuhkan. Ini ditunjukkan dalam contoh berikut, yang memberi Anda akses ke Amazon DynamoDB (DynamoDB).

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";
```

Anda tidak hanya dapat memuat dan menggunakan AWS layanan individual, tetapi Anda juga dapat memuat dan hanya menggunakan perintah layanan yang Anda butuhkan. Ini ditunjukkan dalam contoh berikut, yang memberi Anda akses ke klien DynamoDB dan perintah. `ListTablesCommand`

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

Important

Anda tidak boleh mengimpor submodul ke dalam modul. Misalnya, kode berikut mungkin mengakibatkan kesalahan.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/
CognitoIdentity";
```

Berikut ini adalah kode yang benar.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
```

Membandingkan ukuran kode

Di Versi 2 (V2), contoh kode sederhana yang mencantumkan semua tabel Amazon DynamoDB Anda di `us-west-2` Wilayah mungkin terlihat seperti berikut.

```
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({region: "us-west-2"});
// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

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

```
    } else {
      console.log("Tables names are ", data.TableNames);
    }
  });
```

V3 terlihat seperti berikut ini.

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
(async function () {
  const dbclient = new DynamoDBClient({ region: 'us-west-2'});

  try {
    const results = await dbclient.send(new ListTablesCommand);
    results.TableNames.forEach(function (item, index) {
      console.log(item);
    });
  } catch (err) {
    console.error(err)
  }
})();
```

aws-sdkPaket menambahkan sekitar 40 MB ke aplikasi Anda. Mengganti var AWS = require("aws-sdk") dengan import {DynamoDB} from "@aws-sdk/client-dynamodb" mengurangi overhead itu menjadi sekitar 3 MB. Membatasi impor hanya ke klien ListTablesCommand dan perintah DynamoDB mengurangi overhead menjadi kurang dari 100 KB.

```
// Load the DynamoDB client and ListTablesCommand command for Node.js
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbclient = new DynamoDBClient({});
```

Memanggil perintah di V3

Anda dapat melakukan operasi di V3 menggunakan perintah V2 atau V3. Untuk menggunakan perintah V3 Anda mengimpor perintah dan klien paket AWS Layanan yang diperlukan, dan menjalankan perintah menggunakan .send metode menggunakan pola async/await.

Untuk menggunakan perintah V2, Anda mengimpor paket AWS Layanan yang diperlukan, dan menjalankan perintah V2 secara langsung dalam paket menggunakan pola callback atau `async/await`.

Menggunakan perintah V3

V3 menyediakan serangkaian perintah untuk setiap paket AWS Layanan untuk memungkinkan Anda melakukan operasi untuk AWS Layanan tersebut. Setelah Anda menginstal AWS Layanan, Anda dapat menelusuri perintah yang tersedia di proyek Anda `node_modules/@aws-sdk/client-PACKAGE_NAME/commands` folder.

Anda harus mengimpor perintah yang ingin Anda gunakan. Misalnya, kode berikut memuat layanan DynamoDB, dan perintah `CreateTableCommand`

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

Untuk memanggil perintah ini dalam pola `async/await` yang direkomendasikan, gunakan sintaks berikut.

```
CLIENT.send(new XXXCommand)
```

Misalnya, contoh berikut membuat tabel DynamoDB menggunakan pola `async/await` yang direkomendasikan.

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
const dynamodb = new DynamoDB({region: 'us-west-2'});
var tableParams = {
  Table : TABLE_NAME
};
(async function () => {
  try{
    const data = await dynamodb.send(new CreateTableCommand(tableParams));
    console.log("Success", data);
  }
  catch (err) {
    console.log("Error", err);
  }
})();
```

Menggunakan perintah V2

Untuk menggunakan perintah V2 di SDK for JavaScript, Anda mengimpor paket AWS Layanan lengkap, seperti yang ditunjukkan dalam kode berikut.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

Untuk memanggil perintah V2 dalam pola async/await yang disarankan, gunakan sintaks berikut.

```
client.command(parameters)
```

Contoh berikut menggunakan `createTable` perintah V2 untuk membuat tabel DynamoDB menggunakan pola async/await yang direkomendasikan.

```
const {DynamoDB} = require('@aws-sdk/client-dynamodb');
const dymamoDB = new DynamoDB({region: 'us-west-2'});
var tableParams = {
  TableName : TABLE_NAME
};
async function run() => {
  try {
    const data = await dymamoDB.createTable(tableParams);
    console.log("Success", data);
  }
  catch (err) {
    console.log("Error", err);
  }
};
run();
```

Contoh berikut menggunakan `createBucket` perintah V2 untuk membuat bucket Amazon S3 menggunakan pola callback.

```
const {S3} = require('@aws-sdk/client-s3');
const s3 = new S3({region: 'us-west-2'});
var bucketParams = {
  Bucket : BUCKET_NAME
};
function run(){
  s3.createBucket(bucketParams, function(err, data) {
```

```
        if (err) {
            console.log("Error", err);
        } else {
            console.log("Success", data.Location);
        }
    })
};
```

Tumpukan middleware baru

V2 SDK memungkinkan Anda memodifikasi permintaan selama beberapa tahap siklus hidupnya dengan melampirkan event listener ke permintaan. Pendekatan ini dapat mempersulit debug apa yang salah selama siklus hidup permintaan.

Di V3, Anda dapat menggunakan tumpukan middleware baru untuk mengontrol siklus hidup panggilan operasi. Pendekatan ini memberikan beberapa manfaat. Setiap tahap middleware dalam tumpukan memanggil tahap middleware berikutnya setelah membuat perubahan apa pun pada objek permintaan. Ini juga membuat masalah debugging di tumpukan jauh lebih mudah, karena Anda dapat melihat dengan tepat tahap middleware mana yang dipanggil menjelang kesalahan.

Contoh berikut menambahkan header kustom ke klien Amazon DynamoDB (yang kami buat dan tunjukkan sebelumnya) menggunakan middleware. Argumen pertama adalah fungsi yang menerima `next`, yang merupakan tahap middleware berikutnya dalam tumpukan untuk memanggil, dan `context`, yang merupakan objek yang berisi beberapa informasi tentang operasi yang dipanggil. Fungsi mengembalikan fungsi yang menerima `args`, yang merupakan objek yang berisi parameter yang diteruskan ke operasi dan permintaan. Ia mengembalikan hasil dari memanggil middleware berikutnya dengan `args`.

```
dbclient.middlewareStack.add(
    (next, context) => args => {
        args.request.headers["Custom-Header"] = "value";
        return next(args);
    },
    {
        step: "build"
    }
);

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

Menggunakan SDK dengan Node.js

Node.js adalah runtime lintas platform untuk menjalankan aplikasi sisi server JavaScript . Anda dapat mengatur Node.js di instans Amazon Elastic Compute Cloud (Amazon EC2) 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 kredensial yang diperlukan untuk mengakses layanan web tertentu. Ketika penggunaan API tertentu berbeda antara Node.js dan browser, kami menyebut perbedaan tersebut.

Menggunakan SDK dengan AWS Cloud9

Anda juga dapat mengembangkan aplikasi Node.js menggunakan SDK untuk JavaScript di AWS Cloud9 IDE. Untuk informasi selengkapnya tentang penggunaan AWS Cloud9 dengan SDK for JavaScript, lihat [Gunakan AWS Cloud9 dengan AWS SDK for JavaScript](#).

Menggunakan SDK dengan AWS Amplify

Untuk aplikasi web, seluler, dan hybrid berbasis browser, Anda juga dapat menggunakan [AWS Amplify pustaka](#). GitHub Ini memperluas SDK untuk JavaScript, menyediakan antarmuka deklaratif.

Note

Kerangka kerja seperti Amplify mungkin tidak menawarkan dukungan browser yang sama dengan SDK untuk JavaScript. Lihat dokumentasi kerangka kerja untuk detailnya.

Menggunakan SDK dengan browser web

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

Untuk daftar browser yang didukung oleh AWS SDK for JavaScript, lihat [Browser web yang didukung](#).

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

kredensial yang diperlukan untuk mengakses layanan web tertentu. Ketika penggunaan API tertentu berbeda antara Node.js dan browser, kami menyebut perbedaan tersebut.

Menggunakan browser di V3

V3 memungkinkan Anda untuk menggabungkan dan menyertakan di browser hanya SDK untuk JavaScript file yang Anda butuhkan, mengurangi overhead.

Untuk menggunakan V3 SDK untuk JavaScript di halaman HTML Anda, Anda harus menggabungkan modul klien yang diperlukan dan semua JavaScript fungsi yang diperlukan ke dalam satu JavaScript file menggunakan Webpack, dan menambahkannya dalam tag skrip di halaman HTML Anda.

<head> Misalnya:

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

Note

Untuk informasi selengkapnya tentang Webpack, lihat [Bundel aplikasi dengan webpack](#).

Untuk menggunakan V2 SDK for JavaScript, Anda menambahkan tag skrip yang menunjuk ke versi terbaru SDK V2 sebagai gantinya. Untuk informasi selengkapnya, lihat [contoh](#) di Panduan AWS SDK for JavaScript Pengembang v2.

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 [AWSRepository Contoh Kode](#).

Sumber daya

Selain panduan ini, sumber daya online berikut tersedia untuk SDK untuk JavaScript pengembang:

- [AWS SDK for JavaScriptPanduan Referensi API V3](#)
- [AWSPanduan Referensi SDK dan Alat](#): Berisi pengaturan, fitur, dan konsep dasar lainnya yang umum di antara AWS SDK.
- [JavaScript Blog Pengembang](#)
- [AWS JavaScript Forum](#)
- [JavaScript contoh dalam Katalog AWS Kode](#)
- [AWSRepository Contoh Kode](#)
- [Saluran Gitter](#)
- [Tumpukan Limpah](#)
- [Pertanyaan Stack Overflow TaggeDaws -sdk-js](#)
- GitHub
 - [Sumber SDK](#)
 - [Sumber Dokumentasi](#)

Memulai dengan AWS SDK for JavaScript

AWS SDK for JavaScript ini menyediakan akses ke layanan web baik di browser atau lingkungan Node.js. Bagian ini telah memulai latihan yang menunjukkan kepada Anda cara bekerja dengan SDK untuk JavaScript di setiap JavaScript lingkungan ini.

Note

Anda dapat mengembangkan aplikasi Node.js, dan JavaScript untuk aplikasi berbasis browser, menggunakan SDK untuk JavaScript di IDE. AWS Cloud9 Untuk contoh cara menggunakan AWS Cloud9 pengembangan Node.js, lihat [Gunakan AWS Cloud9 dengan AWS SDK for JavaScript](#).

Topik

- [Otentikasi SDK dengan AWS](#)
- [Memulai dengan Node.js](#)
- [Memulai di browser](#)

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 AWSSDK dan Alat.

Kami merekomendasikan bahwa pengguna baru yang mengembangkan secara lokal dan tidak diberikan metode otentikasi oleh majikan mereka untuk mengatur. 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 [otentikasi IAM Identity Center](#) di AWSSDK dan Tools Reference Guide:

- Itu AWS CLI, yang Anda gunakan untuk memulai sesi portal AWS akses sebelum Anda menjalankan aplikasi Anda.

- [AWSconfigFile bersama](#) yang memiliki [default] profil dengan serangkaian nilai konfigurasi yang dapat direferensikan dari SDK. Untuk menemukan lokasi file ini, lihat [Lokasi file bersama di AWS](#) SDK dan Panduan Referensi Alat.
- configFile bersama menetapkan [region](#) pengaturan. Ini menetapkan default Wilayah AWS 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 kredensial sebelum mengirim permintaan ke. AWS sso_role_nameNilai, yang merupakan peran IAM yang terhubung ke set izin Pusat Identitas IAM, memungkinkan akses ke yang Layanan AWS digunakan dalam aplikasi Anda.

configFile contoh berikut menunjukkan profil default yang diatur dengan konfigurasi penyedia token SSO. sso_sessionPengaturan profil mengacu pada [sso-sessionbagian](#) bernama. sso-sessionBagian 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
```

AWS SDK for JavaScriptV3 tidak memerlukan paket tambahan (seperti SSO danSSO0IDC) untuk ditambahkan ke aplikasi Anda untuk menggunakan autentikasi IAM Identity Center.

Untuk detail tentang penggunaan penyedia kredensi ini secara eksplisit, lihat [fromSSO\(\)](#) di situs web npm (Node.js package manager).

Memulai sesi portal AWS akses

Sebelum menjalankan aplikasi yang mengaksesLayanan 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 diAWS 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 menjalankannya `aws sso login`, Anda tidak akan diminta untuk memberikan kredensial. 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 kredensial jangka pendek, lihat [Kredensial Keamanan Sementara](#) di Panduan Pengguna IAM.
- Untuk mempelajari tentang penyedia kredensi AWS SDK for JavaScript V3 lainnya, lihat Penyedia kredensi [terstandarisasi di Panduan Referensi AWS](#) SDK dan Alat.

Memulai dengan Node.js

Panduan ini menunjukkan cara menginisialisasi paket NPM, menambahkan klien layanan ke paket Anda, dan menggunakan JavaScript SDK untuk memanggil tindakan layanan.

Skenario

Buat paket NPM baru dengan satu file utama yang melakukan hal berikut:

- Membuat bucket Amazon Simple Storage Service
- Menempatkan objek di ember Amazon S3
- Membaca objek di bucket Amazon S3
- Mengonfirmasi jika pengguna ingin menghapus sumber daya

Prasyarat

Sebelum Anda dapat menjalankan contoh, Anda harus melakukan hal berikut:

- Konfigurasi autentikasi SDK Anda. Untuk informasi selengkapnya, lihat [Otentikasi SDK dengan AWS](#).
- Instal [Node.js](#).

Langkah 1: Siapkan struktur paket dan instal paket klien

Untuk mengatur struktur paket dan menginstal paket klien:

1. Buat folder baru `nodegetstarted` untuk berisi paket.
2. Dari baris perintah, navigasikan ke folder baru.
3. Jalankan perintah berikut untuk membuat `package.json` file default:

```
npm init -y
```

4. Jalankan perintah berikut untuk menginstal paket klien Amazon S3:

```
npm i @aws-sdk/client-s3
```

5. Tambahkan "type": "module" ke package.json file. Ini memberitahu Node.js untuk menggunakan sintaks ESM modern. Final package.json akan terlihat mirip dengan yang berikut:

```
{
  "name": "example-javascriptv3-get-started-node",
  "version": "1.0.0",
  "description": "This guide shows you how to initialize an NPM package, add a
service client to your package, and use the JavaScript SDK to call a service
action.",
  "main": "index.js",
  "scripts": {
    "test": "vitest run **/*.unit.test.js"
  },
  "author": "Your Name",
  "license": "Apache-2.0",
  "dependencies": {
    "@aws-sdk/client-s3": "^3.420.0"
  },
  "type": "module"
}
```

Langkah 2: Tambahkan impor dan kode SDK yang diperlukan

Tambahkan kode berikut ke file bernama `index.js` dalam `nodegetstarted` folder.

```
// This is used for getting user input.
import { createInterface } from "readline/promises";

import {
  S3Client,
  PutObjectCommand,
```

```
CreateBucketCommand,  
DeleteObjectCommand,  
DeleteBucketCommand,  
paginateListObjectsV2,  
GetObjectCommand,  
} from "@aws-sdk/client-s3";  
  
export async function main() {  
  // A region and credentials can be declared explicitly. For example  
  // `new S3Client({ region: 'us-east-1', credentials: {...} })` would  
  // initialize the client with those settings. However, the SDK will  
  // use your local configuration and credentials if those properties  
  // are not defined here.  
  const s3Client = new S3Client({});  
  
  // Create an Amazon S3 bucket. The epoch timestamp is appended  
  // to the name to make it unique.  
  const bucketName = `test-bucket-${Date.now()}`;  
  await s3Client.send(  
    new CreateBucketCommand({  
      Bucket: bucketName,  
    })  
  );  
  
  // Put an object into an Amazon S3 bucket.  
  await s3Client.send(  
    new PutObjectCommand({  
      Bucket: bucketName,  
      Key: "my-first-object.txt",  
      Body: "Hello JavaScript SDK!",  
    })  
  );  
  
  // Read the object.  
  const { Body } = await s3Client.send(  
    new GetObjectCommand({  
      Bucket: bucketName,  
      Key: "my-first-object.txt",  
    })  
  );  
  
  console.log(await Body.transformToString());  
  
  // Confirm resource deletion.
```



```
const prompt = createInterface({
  input: process.stdin,
  output: process.stdout,
});

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

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

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

// Call a function if this file was run directly. This allows the file
// to be runnable without running on import.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

Kode contoh dapat ditemukan [di sini GitHub](#).

Langkah 3: Jalankan contoh

Note

Ingatlah untuk masuk! Jika Anda menggunakan IAM Identity Center untuk mengautentikasi, ingatlah untuk masuk menggunakan perintah. AWS CLI `aws sso login`

1. Jalankan `node index.js`.
2. Pilih apakah akan mengosongkan dan menghapus ember.
3. Jika Anda tidak menghapus ember, pastikan untuk mengosongkan dan menghapusnya secara manual nanti.

Memulai di browser

Bagian ini memandu Anda melalui contoh yang menunjukkan cara menjalankan SDK versi 3 (V3) untuk JavaScript di browser.

Note

Menjalankan V3 di browser sedikit berbeda dari versi 2 (V2). Untuk informasi selengkapnya, lihat [Menggunakan browser di V3](#).

Untuk contoh lain penggunaan (V3) SDK for JavaScript, lihat [SDK untuk contoh JavaScript kode \(v3\)](#)

Contoh aplikasi web ini menunjukkan kepada Anda:

- Cara mengakses AWS layanan menggunakan Amazon Cognito untuk otentikasi.
- Cara membaca daftar objek di bucket Amazon Simple Storage Service (Amazon S3) Simple Storage S3) menggunakan AWS Identity and Access Management peran (IAM).

Note

Contoh ini tidak digunakan AWS IAM Identity Center untuk otentikasi.

Skenario

Amazon S3 adalah layanan penyimpanan objek yang menawarkan skalabilitas, ketersediaan data, keamanan, dan kinerja terdepan di industri. Anda dapat menggunakan Amazon S3 untuk menyimpan data sebagai objek dalam wadah yang disebut bucket. Untuk informasi selengkapnya tentang Amazon S3, lihat Panduan Pengguna [Amazon S3](#).

Contoh ini menunjukkan cara menyiapkan dan menjalankan aplikasi web yang mengasumsikan peran IAM untuk dibaca dari bucket Amazon S3. Contoh ini menggunakan pustaka front-end React dan perkakas front-end Vite untuk menyediakan lingkungan pengembangan. JavaScript Aplikasi web menggunakan kumpulan identitas Amazon Cognito untuk memberikan kredensial yang diperlukan untuk mengakses layanan. AWS Contoh kode yang disertakan menunjukkan pola dasar untuk memuat dan menggunakan SDK untuk JavaScript aplikasi web.

Langkah 1: Buat kumpulan identitas Amazon Cognito dan peran IAM

Dalam latihan ini, Anda membuat dan menggunakan kumpulan identitas Amazon Cognito untuk menyediakan akses tidak terautentikasi ke aplikasi web Anda untuk layanan Amazon S3. Membuat kumpulan identitas juga menciptakan peran AWS Identity and Access Management (IAM) untuk mendukung pengguna tamu yang tidak diautentikasi. Untuk contoh 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 dan peran IAM terkait

1. [Masuk ke AWS Management Console dan buka konsol Amazon Cognito di https://console.aws.amazon.com/cognito/](https://console.aws.amazon.com/cognito/).
2. Di panel navigasi kiri, pilih Identity pool.
3. Pilih Buat kumpulan identitas.
4. Di Konfigurasi 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, konfirmasi 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 memerlukan nilai-nilai ini untuk menggantikan `IDENTITY_POOL_ID` dan `REGION` di.* [Langkah 4: Siapkan kode browser](#)

Setelah membuat kumpulan identitas Amazon Cognito, Anda siap menambahkan izin untuk Amazon S3 yang diperlukan oleh aplikasi web Anda.

Langkah 2: Tambahkan kebijakan ke peran IAM yang dibuat

Untuk mengaktifkan akses ke bucket Amazon S3 di aplikasi web Anda, gunakan peran IAM yang tidak diautentikasi (misalnya, `getStartedRole`) yang dibuat untuk kumpulan identitas Amazon Cognito Anda (misalnya, `getStartedPool`). Ini mengharuskan Anda untuk melampirkan 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 S3 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. Pada 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 `ReadOnlyAccessAmazonS3`.

Note

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

6. Pilih Tambahkan izin.

Setelah membuat kumpulan identitas Amazon Cognito dan menambahkan izin untuk Amazon S3 ke peran IAM Anda untuk pengguna yang tidak diautentikasi, Anda siap untuk menambahkan dan mengonfigurasi bucket Amazon S3.

Langkah 3: Tambahkan ember dan objek Amazon S3

Pada langkah ini, Anda akan menambahkan bucket dan objek Amazon S3 untuk contoh. Anda juga akan mengaktifkan cross-origin resource sharing (CORS) untuk bucket. Untuk informasi selengkapnya tentang membuat bucket dan objek Amazon S3, lihat [Memulai Amazon S3 di Panduan Pengguna Amazon S3](#).

Untuk menambahkan bucket dan objek Amazon S3 dengan CORS

1. Masuk ke AWS Management Console dan buka konsol Amazon S3 di <https://console.aws.amazon.com/s3/>.
2. Di panel navigasi kiri, pilih Bucket dan pilih Buat ember.
3. Masukkan nama bucket yang sesuai dengan [aturan penamaan bucket](#) (misalnya, getstartedbucket) dan pilih Create bucket.
4. Pilih bucket yang Anda buat, lalu pilih tab Objects. Kemudian pilih Unggah.
5. Di Bawah File dan folder, pilih Tambahkan berkas.
6. Pilih file yang akan diunggah, lalu pilih Buka. Kemudian pilih Unggah untuk menyelesaikan pengunggahan objek ke bucket Anda.
7. Selanjutnya, pilih tab Izin di bucket Anda, lalu pilih Edit di bagian Cross-origin resource sharing (CORS). Masukkan JSON berikut:

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

```
]
```

8. Pilih Simpan perubahan.

Setelah Anda menambahkan bucket Amazon S3 dan menambahkan objek, Anda siap untuk mengatur kode browser.

Langkah 4: Siapkan kode browser

Contoh aplikasi terdiri dari aplikasi React satu halaman. File untuk contoh ini dapat ditemukan [di sini](#) [GitHub](#).

Untuk mengatur aplikasi contoh

1. Instal [Node.js](#).
2. Dari baris perintah, kloning [Repositori Contoh AWS Kode](#):

```
git clone --depth 1 https://github.com/awsdocs/aws-doc-sdk-examples.git
```

3. Arahkan ke aplikasi contoh:

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

4. Jalankan perintah berikut untuk menginstal paket yang diperlukan:

```
npm install
```

5. Selanjutnya, buka `src/App.tsx` di editor teks dan lengkapi yang berikut ini:
 - Ganti *YOUR_IDENTITY_POOL_ID* dengan *ID kumpulan* identitas Amazon Cognito yang Anda catat. [Langkah 1: Buat kumpulan identitas Amazon Cognito dan peran IAM](#)
 - Ganti nilai untuk wilayah ke wilayah yang ditetapkan untuk bucket Amazon S3 dan kumpulan identitas Amazon Cognito. Perhatikan bahwa wilayah untuk kedua layanan harus sama (misalnya, `us-east-2`).
 - Ganti *bucket-name* dengan *nama* bucket yang Anda buat. [Langkah 3: Tambahkan ember dan objek Amazon S3](#)

Setelah Anda mengganti teks, simpan `App.tsx` file. Anda sekarang siap untuk menjalankan aplikasi web.

Langkah 5: Jalankan Contoh

Untuk menjalankan aplikasi contoh

1. Dari baris perintah, arahkan ke aplikasi contoh:

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

2. Dari baris perintah, jalankan perintah berikut:

```
npm run dev
```

Lingkungan pengembangan Vite akan berjalan dengan pesan berikut:

```
VITE v4.3.9 ready in 280 ms

# Local:   http://localhost:5173/
# Network: use --host to expose
# press h to show help
```

3. Di browser web Anda, navigasikan ke URL yang ditunjukkan di atas (misalnya, <http://localhost:5173/>). Aplikasi contoh akan menampilkan daftar nama file objek di bucket Amazon S3 Anda.

Pembersihan

Untuk membersihkan sumber daya yang Anda buat selama tutorial ini, lakukan hal berikut:

- [Di konsol Amazon S3](#), hapus objek dan bucket apa pun yang dibuat (misalnya, `getstartedbucket`).
- [Di konsol IAM](#), hapus nama peran (misalnya, `getStartedRole`).
- [Di konsol Amazon Cognito](#), hapus nama kumpulan identitas (misalnya, `getStartedPool`).

Siapkan SDK untuk JavaScript

Topik di bagian ini menjelaskan cara menginstal dan memuat SDK JavaScript agar 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-native](#) arsip untuk detailnya.

Topik

- [Prasyarat](#)
- [Instal SDK untuk JavaScript](#)
- [Muat SDK untuk JavaScript](#)

Prasyarat

Instal Node.js di server Anda, jika belum diinstal.

Topik

- [Menyiapkan lingkungan AWS Node.js](#)
- [Browser web yang 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. Kemudian buat instans Amazon EC2 menggunakan AMI itu. Saat membuat instans Amazon EC2 Anda, pilih AMI Anda dari. AWS Marketplace Cari Node.js dan pilih opsi AMI yang menyertakan versi Node.js yang sudah diinstal sebelumnya (32-bit atau 64-bit).

- Buat instans Amazon EC2 dan instal Node.js di atasnya. Untuk informasi selengkapnya tentang cara menginstal Node.js pada instans Amazon Linux, lihat [Menyiapkan Node.js pada instans Amazon EC2](#).
- 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.
- Buat server aplikasi Node.js menggunakan AWS OpsWorks. Untuk informasi selengkapnya tentang menggunakan Node.js dengan AWS OpsWorks, lihat [Membuat tumpukan Node.js pertama Anda](#) di Panduan AWS OpsWorks Pengguna.

Browser web yang didukung

AWS SDK for JavaScript Mendukung semua browser web modern.

Di versi 3.183.0 atau yang lebih baru, SDK untuk JavaScript menggunakan artefak ES2020, yang mendukung versi minimum berikut.

Peramban	Versi
Google Chrome	80,0+
Mozilla Firefox	80,0+
Opera	63.0+
Microsoft Edge	80,0+
Apple Safari	14.1+
Samsung Internet	12.0+

Di versi 3.182.0 atau sebelumnya, SDK untuk JavaScript menggunakan artefak ES5, yang mendukung versi minimum berikut.

Peramban	Versi
Google Chrome	49.0+
Mozilla Firefox	45.0+
Opera	36.0+
Microsoft Edge	12.0+
Penjelajah Internet Windows	N/A
Apple Safari	9.0+
Peramban Android	76.0+
Peramban UC	12.12+
Samsung Internet	5.0+

Note

Kerangka kerja seperti AWS Amplify mungkin tidak menawarkan dukungan browser yang sama dengan SDK untuk JavaScript. Lihat [AWS Amplify Dokumentasi](#) untuk detailnya.

Instal SDK untuk JavaScript

Tidak semua layanan segera tersedia di SDK atau di semua AWS Wilayah.

Untuk menginstal layanan dari AWS SDK for JavaScript menggunakan [npm, manajer paket Node.js](#), masukkan perintah berikut pada prompt perintah, di mana **SERVICE** adalah nama layanan, seperti `3`.

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

Untuk daftar lengkap paket klien AWS SDK for JavaScript layanan, lihat [panduan Referensi AWS SDK for JavaScript API](#).

Muat SDK untuk JavaScript

Setelah Anda menginstal SDK, Anda dapat memuat paket klien dalam aplikasi node Anda menggunakan `import`. Misalnya, untuk memuat klien Amazon S3 dan perintah Amazon [ListBucketsS3](#), gunakan yang berikut ini.

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
```

Konfigurasi SDK untuk JavaScript

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

- AWS Wilayah di mana Anda akan meminta layanan
- Bagaimana kode Anda mengautentikasi dengan AWS

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.

[AWS SDK dan Tools Reference Guide](#) juga berisi pengaturan, fitur, dan konsep dasar lainnya yang umum di antara banyak SDK AWS .

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

Topik

- [Konfigurasi per layanan](#)
- [Mengatur AWS Wilayah](#)
- [Tetapkan kredensialnya](#)
- [Pertimbangan Node.js](#)
- [Pertimbangan Skrip Browser](#)

Konfigurasi per layanan

Anda dapat mengonfigurasi SDK dengan meneruskan informasi konfigurasi ke objek layanan.

Konfigurasi tingkat layanan memberikan kontrol yang signifikan atas layanan individual, memungkinkan Anda untuk memperbarui konfigurasi objek layanan individual ketika kebutuhan Anda bervariasi dari konfigurasi default.

Note

Dalam versi 2.x konfigurasi AWS SDK for JavaScript layanan dapat diteruskan ke konstruktor klien individu. Namun, konfigurasi ini pertama-tama akan digabungkan secara otomatis ke dalam salinan konfigurasi SDK global. `AWS.config`

Selain itu, `AWS.config.update({/* params */})` hanya memanggil konfigurasi yang diperbarui untuk klien layanan yang dipakai setelah panggilan pembaruan dilakukan, bukan klien yang ada.

Perilaku ini sering menjadi sumber kebingungan, dan membuatnya sulit untuk menambahkan konfigurasi ke objek global yang hanya memengaruhi sebagian klien layanan dengan cara yang kompatibel ke depan. Di versi 3, tidak ada lagi konfigurasi global yang dikelola oleh SDK. Konfigurasi harus diteruskan ke setiap klien layanan yang dipakai. Masih dimungkinkan untuk berbagi konfigurasi yang sama di beberapa klien tetapi konfigurasi itu tidak akan digabungkan secara otomatis dengan status global.

Tetapkan 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.

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

```
var ec2_regionA = new EC2({region: 'ap-southeast-2', maxAttempts: 15});
var ec2_regionB = new EC2({region: 'us-west-2', maxAttempts: 15});
```

Mengatur AWS Wilayah

AWS 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 Region saat membuat klien layanan di SDK JavaScript agar SDK mengakses layanan di Wilayah tersebut. Beberapa layanan hanya tersedia di Wilayah tertentu.

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

Dalam konstruktor kelas klien

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

```
const s3Client = new S3.S3Client({region: 'us-west-2'});
```

Gunakan variabel lingkungan

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

Gunakan file konfigurasi bersama

Sama seperti file kredensial bersama yang memungkinkan Anda menyimpan kredensial untuk digunakan oleh SDK, Anda dapat menyimpan AWS Region dan pengaturan konfigurasi lainnya dalam file bersama yang diberi nama `config` untuk digunakan SDK. Jika variabel `AWS_SDK_LOAD_CONFIG` lingkungan disetel ke nilai kebenaran, 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-west-2
  output=json
```

Untuk informasi selengkapnya tentang menggunakan file bersama config dan credentials file, lihat File [konfigurasi dan kredensial bersama di Panduan Referensi AWS](#) SDK dan Alat.

Urutan prioritas untuk pengaturan Wilayah

Berikut ini adalah urutan prioritas untuk pengaturan Wilayah:

1. Jika Region diteruskan ke konstruktor kelas klien, Region itu digunakan.
2. Jika Region diatur dalam variabel lingkungan, Region yang digunakan.
3. Jika tidak, Wilayah yang ditentukan dalam file konfigurasi bersama akan digunakan.

Tetapkan kredensialnya

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 per layanan, dengan meneruskan kredensial langsung ke objek layanan.

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

Praktik terbaik untuk kredensial

Menetapkan kredensial 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 kredensial 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 terkecil](#) dari topik Praktik Terbaik di Panduan Pengguna IAM.

Topik

- [Setel kredensial di Node.js](#)
- [Mengatur kredensial di browser web](#)

Setel kredensial di Node.js

Kami merekomendasikan bahwa pengguna baru yang mengembangkan secara lokal dan tidak diberikan metode otentikasi oleh majikan mereka untuk mengatur. AWS IAM Identity Center Untuk informasi selengkapnya, lihat [Otentikasi SDK dengan AWS](#).

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 kredensial 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.

AWS SDK for JavaScript V3 menyediakan rantai penyedia kredensial default di Node.js, sehingga Anda tidak diharuskan untuk menyediakan penyedia kredensial secara eksplisit. [Rantai penyedia kredensial](#) default mencoba menyelesaikan kredensial dari berbagai sumber berbeda dalam prioritas tertentu, hingga kredensial dikembalikan dari salah satu sumber. [Anda dapat menemukan rantai penyedia kredensial untuk SDK untuk JavaScript V3 di sini](#).

Rantai penyedia kredensial

Semua SDK memiliki serangkaian tempat (atau sumber) yang mereka periksa untuk mendapatkan kredensial yang valid untuk digunakan untuk membuat permintaan ke file. Layanan AWS Setelah kredensial yang valid ditemukan, pencarian dihentikan. Pencarian sistematis ini disebut rantai penyedia kredensial default.

Untuk setiap langkah dalam rantai, ada berbagai cara untuk mengatur nilai. Menetapkan nilai secara langsung dalam kode selalu diutamakan, diikuti dengan pengaturan sebagai variabel lingkungan, dan kemudian di file bersama AWS `config`. Untuk informasi selengkapnya, [lihat Prioritas pengaturan di AWS SDK dan Panduan Referensi Alat](#).

AWS SDK dan Tools Reference Guide memiliki informasi tentang pengaturan konfigurasi SDK yang digunakan oleh semua AWS SDK dan. AWS CLI Untuk mempelajari lebih lanjut tentang cara mengonfigurasi SDK melalui AWS `config` file bersama, lihat File [konfigurasi dan kredensial](#)

[bersama](#). Untuk mempelajari lebih lanjut tentang cara mengonfigurasi SDK melalui pengaturan variabel lingkungan, lihat [Dukungan variabel lingkungan](#).

Untuk mengautentikasi dengan AWS, AWS SDK for JavaScript memeriksa penyedia kredensi dalam urutan yang tercantum dalam tabel berikut.

AWS SDK for JavaScript Metode penyedia kredensi Referensi API berdasarkan prioritas	Penyedia kredensi tersedia	AWS Panduan Referensi SDK dan Alat
fromEnv()	AWS kunci akses dari variabel lingkungan	AWS kunci akses
fromSSO()	AWS IAM Identity Center. Dalam panduan ini, lihat Otentikasi SDK dengan AWS .	Penyedia kredensi Pusat Identitas IAM
fromIni()	AWS kunci akses dari berbagai config dan credentials file	AWS kunci akses
	Penyedia entitas tepercaya (seperti <code>AWS_ROLE_ARN</code>)	Asumsikan peran IAM
	Token identitas web dari AWS Security Token Service (AWS STS)	Bersekutu dengan identitas web atau OpenID Connect
	Kredensi Amazon Elastic Container Service (Amazon ECS)	Penyedia kredensi kontainer
	Kredensi profil instans Amazon Elastic Compute Cloud (Amazon EC2) (penyedia kredensi IMDS)	Penyedia kredensi IMDS

AWS SDK for JavaScript Metode penyedia kredensi Referensi API berdasarkan prioritas	Penyedia kredensi tersedia	AWS Panduan Referensi SDK dan Alat
	Penyedia kredensi proses	Penyedia kredensi proses
	AWS IAM Identity Center kredensialnya	Penyedia kredensi Pusat Identitas IAM
fromProcess()	Penyedia kredensi proses	Penyedia kredensi proses
fromTokenFile()	Token identitas web dari AWS Security Token Service (AWS STS)	Bersekutu dengan identitas web atau OpenID Connect
fromContainerMetad ata()	Kredensi Amazon Elastic Container Service (Amazon ECS)	Penyedia kredensi kontainer
fromInstanceMetada ta()	Kredensi profil instans Amazon Elastic Compute Cloud (Amazon EC2) (penyedia kredensi IMDS)	Penyedia kredensi IMDS

Jika Anda mengikuti pendekatan yang disarankan bagi pengguna baru untuk memulai, Anda menyiapkan AWS IAM Identity Center [Otentikasi SDK dengan AWS](#) autentikasi selama topik Memulai. Metode otentikasi lainnya berguna untuk situasi yang berbeda. Untuk menghindari risiko keamanan, kami sarankan untuk selalu menggunakan kredensi jangka pendek. Untuk prosedur metode otentikasi lainnya, lihat [Otentikasi dan akses di Panduan](#) Referensi AWS SDK dan Alat.

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

Topik

- [Memuat kredensi di Node.js dari peran IAM untuk Amazon EC2](#)
- [Memuat kredensi untuk fungsi Lambda Node.js](#)

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

Jika menjalankan aplikasi Node.js di instans Amazon EC2, 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 kredensial IAM untuk aplikasi Anda, sehingga tidak perlu menyediakan kredensial secara manual.

Untuk informasi selengkapnya tentang menambahkan peran IAM ke instans Amazon EC2, [lihat peran IAM untuk Amazon EC2](#).

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 kredensial 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

Mengatur kredensial di browser web

Ada beberapa cara untuk memasok kredensial 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 kredensial Anda, dalam urutan rekomendasi:

1. Menggunakan Identitas Amazon Cognito untuk mengautentikasi pengguna dan menyediakan kredensial
2. Menggunakan identitas federasi web

⚠ Warning

Kami tidak menyarankan hard coding AWS kredensial Anda dalam skrip Anda. Kredensial pengkodean keras menimbulkan risiko mengekspos ID kunci akses dan kunci akses rahasia Anda.

Topik

- [Menggunakan Identitas Amazon Cognito untuk mengautentikasi pengguna](#)

Menggunakan Identitas Amazon Cognito untuk mengautentikasi pengguna

Cara yang disarankan untuk mendapatkan AWS kredensial untuk skrip browser Anda adalah dengan menggunakan klien kredensial Identitas Amazon Cognito. `CognitoIdentityClient` 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 kredensial menggunakan dukungan federasi identitas web di AWS Security Token Service (`AWSSecurityTokenService`).

Amazon Cognito membantu Anda mengelola abstraksi identitas di beberapa penyedia identitas. Identitas yang dimuat kemudian ditukar dengan kredensial di `AWSSecurityTokenService`.

Konfigurasi 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 klien Amazon Cognito](#) Anda. Buat dan kaitkan peran IAM yang diautentikasi dan tidak diautentikasi untuk kumpulan identitas Anda. Untuk informasi selengkapnya, lihat [Tutorial: Membuat kumpulan identitas](#) di Panduan Pengembang Amazon Cognito.

Pengguna yang tidak diautentikasi tidak memverifikasi identitasnya, membuat peran ini sesuai untuk pengguna tamu aplikasi Anda atau jika tidak masalah jika pengguna telah memverifikasi identitasnya. 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 Anda mengonfigurasi kumpulan identitas, gunakan `fromCognitoIdentityPool` metode dari `@aws-sdk/credential-providers` untuk mengambil kredensial dari kumpulan identitas. Dalam contoh berikut untuk membuat klien Amazon S3, ganti *AWS_REGION* dengan *region* dan *IDENTITY_POOL_ID* dengan *ID kumpulan identitas*.

```
// Import required AWS SDK clients and command for Node.js
import {S3Client} from "@aws-sdk/client-s3";
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";

const REGION = AWS_REGION;

const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
    CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: {
      // Optional tokens, used for authenticated login.
    },
  })
});
```

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 Anda menggunakan kumpulan pengguna Amazon Cognito sebagai penyedia otentikasi, Anda dapat menggunakan metode yang mirip dengan yang di bawah ini.

```
// Get the Amazon Cognito ID token for the user. 'getToken()' below.
let idToken = getToken();
let COGNITO_ID = "COGNITO_ID"; // 'COGNITO_ID' has the format 'cognito-
idp.REGION.amazonaws.com/COGNITO_USER_POOL_ID'
let loginData = {
  [COGNITO_ID]: idToken,
};
const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
    CognitoIdentityClient.
```

```
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: loginData
  })
});

// Strips the token ID from the URL after authentication.
window.getToken = function () {
  var idtoken = window.location.href;
  var idtoken1 = idtoken.split("=")[1];
  var idtoken2 = idtoken1.split("&")[0];
  var idtoken3 = idtoken2.split("&")[0];
  return idtoken3;
};
```

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 hal ini, kredensial default Anda mungkin terlihat seperti berikut:

```
// Import the required AWS SDK for JavaScript v3 modules.
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";
// Set the default credentials.
const creds = new fromCognitoIdentityPool({
  IdentityPoolId: "IDENTITY_POOL_ID",
  clientConfig({ region: REGION }) // Configure the underlying CognitoIdentityClient.
});
```

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 khusus yang memperbarui objek kredensial 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;
}
```

Pertimbangan Node.js

Meskipun kode Node.js adalah JavaScript, menggunakan AWS SDK for 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 API tergantung pada keakraban Anda dengan pola pengkodean Node.js umum, seperti mengimpor dan menggunakan modul Node.js lainnya seperti modul. File System (`fs`)

Gunakan modul Node.js bawaan

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.

```
import http from 'http';
```

Memanggil metode modul seolah-olah mereka adalah metode dari objek itu. Misalnya, berikut adalah kode yang membaca file HTML.

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

Untuk daftar lengkap semua modul bawaan yang disediakan Node.js, lihat [dokumentasi Node.js](#) di situs web Node.js.

Gunakan 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](#).

Konfigurasi 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 bagaimana mengatur `maxSockets` untuk klien DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import https from "https";
let agent = new https.Agent({
  maxSockets: 25
});

let dynamoDbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    requestTimeout: 3_000,
    httpsAgent: agent
  });
});
```

SDK untuk JavaScript menggunakan `maxSockets` nilai 50 jika Anda tidak memberikan nilai atau `Agent` objek. Jika Anda menyediakan `Agent` objek, `maxSockets` nilainya akan digunakan. Untuk informasi selengkapnya tentang pengaturan `maxSockets` di Node.js, lihat [dokumentasi Node.js](#).

Pada v3.521.0 dari AWS SDK for JavaScript, Anda dapat menggunakan sintaks [singkatan](#) berikut untuk mengkonfigurasi `requestHandler`

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

const client = new DynamoDBClient({
  requestHandler: {
    requestTimeout: 3_000,
    httpsAgent: { maxSockets: 25 },
  },
});
```

Gunakan kembali koneksi dengan keep-alive di Node.js

Agan HTTP/HTTPS Node.js default membuat koneksi TCP baru untuk setiap permintaan baru. Untuk menghindari biaya pembuatan koneksi baru, SDK untuk JavaScript menggunakan kembali koneksi TCP.

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

Anda juga dapat menonaktifkan menjaga koneksi ini tetap hidup pada basis klien per layanan, seperti yang ditunjukkan dalam contoh berikut untuk klien DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "http";

const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: new Agent({ keepAlive: false })
  })
});
```

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

Konfigurasi 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.

Untuk menemukan agen HTTP pihak ketiga, cari “HTTP proxy” di [npm](#).

Untuk menginstal proxy agen HTTP pihak ketiga, masukkan yang berikut ini di command prompt, di mana *PROXY* adalah nama npm paket.

```
npm install PROXY --save
```

Untuk menggunakan proxy dalam aplikasi Anda, gunakan `httpsAgent` properti `httpAgent` and, seperti yang ditunjukkan pada contoh berikut untuk klien DynamoDB.

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { NodeHttpHandler } from '@smithy/node-http-handler';
import { HttpsProxyAgent } from 'hpagent';
const agent = new HttpsProxyAgent({ proxy: "http://internal.proxy.com" });
const dynamoDbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  }),
});
```

Note

`httpAgent` tidak sama dengan `httpsAgent`, dan karena sebagian besar panggilan dari klien akan ke `https`, keduanya harus disetel.

Daftarkan 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 dibuat kemudian `https.Agent` digunakan oleh klien DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";
import { readFileSync } from "fs";
const certs = [readFileSync("/path/to/cert.pem")];
const agent = new Agent({
  rejectUnauthorized: true,
  ca: certs
});
const dynamoDbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  })
});
```

Pertimbangan Skrip Browser

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

Topik

- [Membangun SDK untuk Browser](#)
- [Cross-origin resource sharing \(CORS\)](#)
- [Bundel aplikasi dengan webpack](#)

Membangun SDK untuk Browser

Tidak seperti SDK untuk JavaScript versi 2 (V2), V3 tidak disediakan sebagai JavaScript file dengan dukungan yang disertakan untuk serangkaian layanan default. Sebagai gantinya, V3 memungkinkan Anda untuk menggabungkan dan menyertakan di browser hanya SDK untuk JavaScript file yang Anda butuhkan, mengurangi overhead. Sebaiknya gunakan Webpack untuk menggabungkan SDK yang diperlukan untuk JavaScript file, dan paket pihak ketiga tambahan yang Anda butuhkan, ke dalam satu Javascript file, dan memuatnya ke dalam skrip browser menggunakan tag `<script>`

Untuk informasi selengkapnya tentang Webpack, lihat [Bundel aplikasi dengan webpack](#). Untuk contoh yang menggunakan Webpack untuk memuat V3 SDK JavaScript ke dalam browser, lihat [Membangun aplikasi untuk mengirimkan data ke DynamoDB](#)

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.

Menggunakan SDK Builder untuk membangun SDK JavaScript

Note

Amazon Web Services versi 3 (V3) tidak lagi mendukung Browser Builder. Untuk meminimalkan penggunaan bandwidth aplikasi browser, kami sarankan Anda mengimpor modul bernama, dan bundel mereka untuk mengurangi ukuran. Untuk informasi selengkapnya tentang bundling, lihat [Bundel aplikasi dengan webpack](#).

Cross-origin resource sharing (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 for 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.

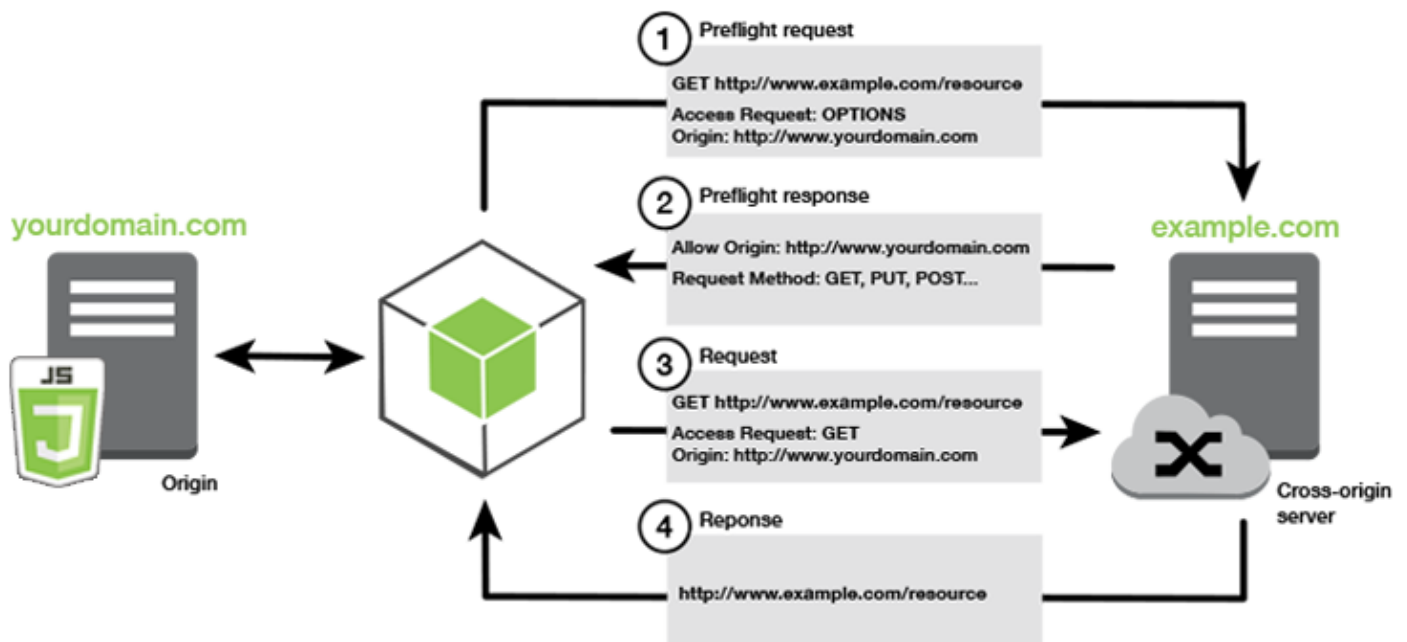
CORS menentukan apakah akan mengizinkan pembagian sumber daya dalam permintaan lintas asal berdasarkan hal berikut:

- 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.

Konfigurasi CORS untuk bucket Amazon S3

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

Jika Anda mengonfigurasi CORS di Konsol Manajemen Layanan AWS Web, Anda harus menggunakan JSON untuk membuat konfigurasi CORS. Konsol Manajemen Layanan AWS Web baru hanya mendukung konfigurasi JSON CORS.

Important

Di Konsol Manajemen Layanan AWS Web yang baru, konfigurasi CORS harus JSON.

1. Di Konsol Manajemen Layanan AWS Web, buka konsol Amazon S3, temukan bucket yang ingin Anda konfigurasi dan pilih kotak centang.
2. Di panel yang terbuka, pilih Izin.
3. Pada tab Izin, pilih Konfigurasi CORS.
4. Masukkan konfigurasi CORS Anda di CORS Configuration Editor, lalu pilih Simpan.

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

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

Misalnya 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 `domainexample.org`. Namun, kami menyarankan Anda `<AllowedOrigin>` untuk menjangkau 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, membaca `ETag` header dari upload 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-`, seperti `x-amz-meta-my-custom-header`, dan juga harus diekspos dengan cara yang sama.

Bundel aplikasi dengan webpack

Penggunaan modul kode oleh aplikasi web dalam skrip browser atau 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`

`webpack` bundler mem-parsing kode aplikasi Anda, mencari `import` atau `require` pernyataan, untuk membuat bundel yang berisi semua aset yang dibutuhkan aplikasi Anda. Ini agar aset dapat dengan mudah dilayani melalui halaman web. SDK for JavaScript dapat dimasukkan `webpack` sebagai salah satu dependensi untuk disertakan dalam bundel keluaran.

Untuk informasi selengkapnyawebpack, lihat [bundler modul webpack](#) aktif. GitHub

Instal webpack

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

```
npm install --save-dev webpack
```

Untuk menggunakan `path` modul untuk bekerja dengan jalur file dan direktori, yang diinstal secara otomatis dengan `webpack`, Anda mungkin perlu menginstal `path-browserify` paket Node.js.


```
npm install --save-dev path-browserify
```

Konfigurasi webpack

Secara default, Webpack mencari JavaScript file bernama `webpack.config.js` di direktori root proyek Anda. File ini menentukan opsi konfigurasi Anda. Berikut ini adalah contoh file `webpack.config.js` konfigurasi untuk WebPack versi 5.0.0 dan yang lebih baru.

Note

Persyaratan konfigurasi webpack bervariasi tergantung pada versi Webpack yang Anda instal. Untuk informasi selengkapnya, lihat [dokumentasi Webpack](#).

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Enable WebPack to use the 'path' package.
  resolve: {
    fallback: { path: require.resolve("path-browserify")}
  }
  /**
   * In Webpack version v2.0.0 and earlier, you must tell
   * webpack how to use "json-loader" to load 'json' files.
   * To do this Enter 'npm --save-dev install json-loader' at the
   * command line to install the "json-loader" package, and include the
   * following entry in your webpack.config.js.
   * module: {
   *   rules: [{test: /\.json$/, use: use: "json-loader"}]
   * }
   */
};
```

Dalam contoh ini, `browser.js` ditentukan sebagai titik masuk. Titik masuk adalah file yang webpack digunakan 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.

Jalankan webpack

Untuk membangun aplikasi yang akan digunakan webpack, tambahkan berikut ini ke `scripts` objek dalam `package.json` file Anda.

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

Berikut ini adalah contoh `package.json` file 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/client-iam": "^3.32.0",
    "@aws-sdk/client-s3": "^3.32.0"
  },
  "devDependencies": {
    "webpack": "^5.0.0"
  }
}
```

Untuk membangun aplikasi Anda, masukkan perintah berikut.

```
npm run build
```

webpackModul bundler kemudian menghasilkan JavaScript file yang Anda tentukan di direktori root proyek Anda.

Gunakan bundel webpack

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

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

Bundel untuk Node.js

Anda dapat menggunakan webpack untuk menghasilkan bundel yang berjalan di Node.js dengan menentukan `node` 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, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Let webpack know to generate a Node.js bundle.
  target: "node",
  // Enable WebPack to use the 'path' package.
  resolve: {
    fallback: { path: require.resolve("path-browserify")}
  }
}
```

```
/**
 * In Webpack version v2.0.0 and earlier, you must tell
 * webpack how to use "json-loader" to load 'json' files.
 * To do this Enter 'npm --save-dev install json-loader' at the
 * command line to install the "json-loader" package, and include the
 * following entry in your webpack.config.js.
module: {
  rules: [{test: /\.json$/, use: use: "json-loader"}]
}
**/
};
```

Bekerja dengan AWS layanan di SDK untuk JavaScript

AWS SDK for JavaScript V3 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 API tingkat rendah untuk menggunakan fitur dan sumber daya layanan. Misalnya, Amazon DynamoDB API tersedia melalui kelas. `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 berisi nol atau lebih properti sebagai parameter JSON. Respons dikapsulasi dalam objek yang terkait dengan operasi, dan dikembalikan ke pemohon melalui salah satu dari beberapa teknik, seperti fungsi callback atau janji. JavaScript

Topik

- [Membuat dan memanggil objek layanan](#)
- [Layanan panggilan secara asinkron](#)
- [Buat permintaan klien layanan](#)
- [Menangani tanggapan klien layanan](#)
- [Bekerja dengan JSON](#)
- [SDK untuk contoh JavaScript kode](#)

Membuat dan memanggil objek layanan

JavaScript API mendukung sebagian besar AWS layanan yang tersedia. Setiap layanan di JavaScript API menyediakan kelas klien dengan `send` metode yang Anda gunakan untuk memanggil setiap API yang didukung layanan. 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 untuk setiap layanan yang Anda butuhkan ke aplikasi Anda menggunakan `import`, yang menyediakan dukungan untuk semua layanan saat ini. Contoh berikut membuat objek layanan Amazon S3 di Wilayah. `us-west-1`

```
// Import the Amazon S3 service client
import { S3Client } from "@aws-sdk/client-s3";
// Create an S3 client in the us-west-1 Region
const s3Client = new S3Client({
  region: "us-west-1"
});
```

Tentukan 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 `GetObjectCommand` metode dari `S3Client`. Untuk informasi selengkapnya tentang meneruskan parameter JSON, lihat [Bekerja dengan JSON](#).

```
s3Client.send(new GetObjectCommand({Bucket: 'bucketName', Key: 'keyName'}));
```

Untuk informasi selengkapnya tentang parameter Amazon S3, lihat [@aws-sdk/client-s3](#) di Referensi API.

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.

Meskipun Anda dapat menggunakan salah satu teknik ini untuk mengelola panggilan asinkron, kami menyarankan Anda menggunakan `async/await` untuk semua kode baru.

async/menunggu

Kami menyarankan Anda menggunakan teknik ini karena ini adalah perilaku default di V3.

janji

Gunakan teknik ini di browser yang tidak mendukung async/await.

panggilan balik

Hindari menggunakan callback kecuali dalam kasus yang sangat sederhana. Namun, Anda mungkin merasa berguna untuk skenario migrasi.

Topik

- [Kelola panggilan asinkron](#)
- [Gunakan async/await](#)
- [Gunakan JavaScript janji](#)
- [Gunakan fungsi callback anonim](#)

Kelola panggilan asinkron

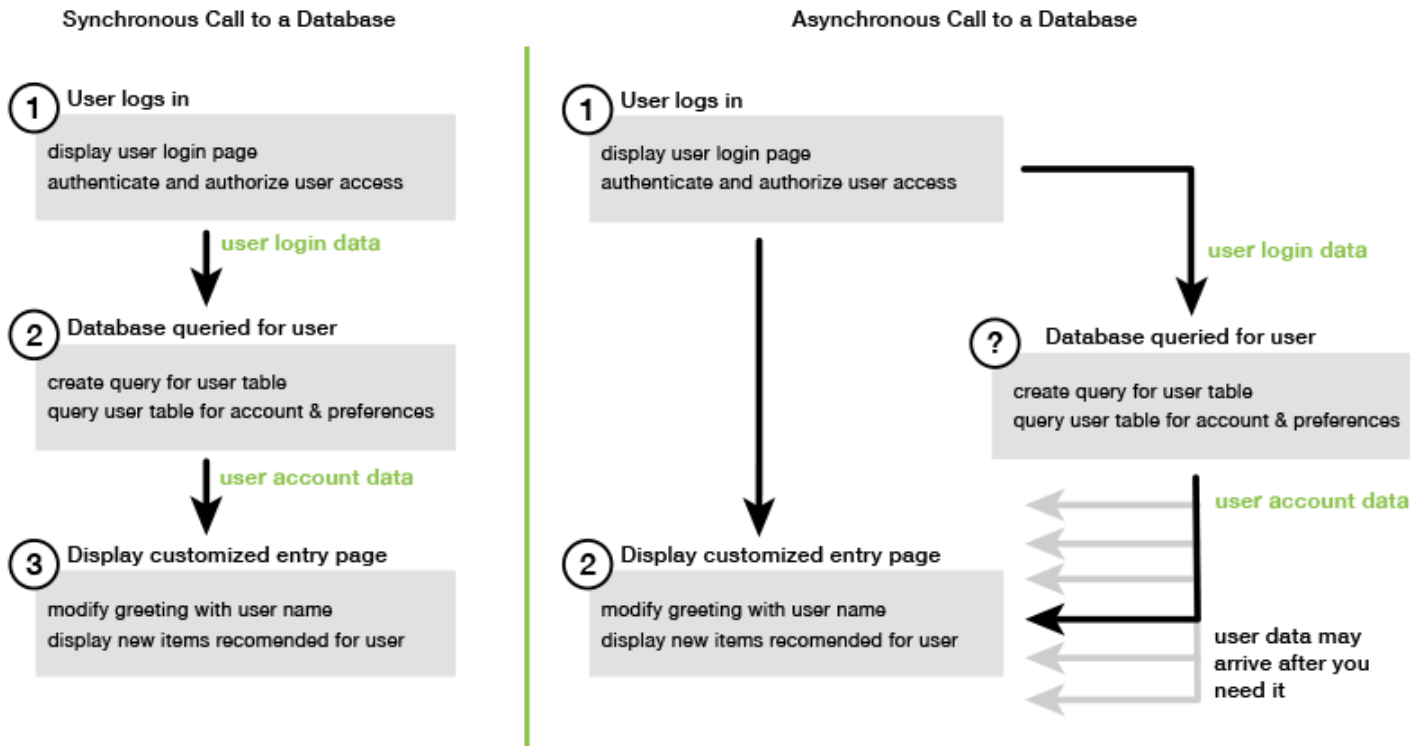
Misalnya, halaman beranda situs web e-commerce memungkinkan pelanggan yang kembali masuk. Bagian dari manfaat bagi pelanggan yang masuk adalah bahwa, setelah masuk, situs kemudian menyesuaikan diri dengan preferensi khusus mereka. Untuk membuat ini terjadi:

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 asinkron. 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.



Gunakan async/await

Daripada menggunakan janji, Anda harus mempertimbangkan untuk menggunakan `async/await`. Fungsi asinkron lebih sederhana dan membutuhkan lebih sedikit boilerplate daripada menggunakan janji. `await` hanya dapat digunakan dalam fungsi `async` untuk menunggu nilai secara asinkron.

Contoh berikut menggunakan `async/await` untuk mencantumkan semua tabel Amazon DynamoDB Anda. `us-west-2`

i Note

Untuk contoh ini untuk menjalankan:

- Instal klien AWS SDK for JavaScript DynamoDB dengan `npm install @aws-sdk/client-dynamodb` masukkan baris perintah proyek Anda.
- Pastikan Anda telah mengonfigurasi AWS kredensialnya dengan benar. Untuk informasi selengkapnya, lihat [Tetapkan kredensialnya](#).


```
import { DynamoDBClient,
ListTablesCommand } from "@aws-sdk/client-dynamodb";
(async function () {
  const dbClient = new DynamoDBClient({ region: "us-west-2" });
  const command = new ListTablesCommand({});

  try {
    const results = await dbClient.send(command);
    console.log(results.TableNames.join('\n'));
  } catch (err) {
    console.error(err)
  }
})();
```

Note

Tidak semua browser mendukung async/await. Lihat [Fungsi async](#) untuk daftar browser dengan dukungan async/await.

Gunakan JavaScript janji

Gunakan metode AWS SDK for JavaScript v3 klien layanan (`ListTablesCommand`) untuk membuat panggilan layanan dan mengelola aliran asinkron alih-alih menggunakan panggilan balik. Contoh berikut menunjukkan cara mendapatkan nama tabel Amazon DynamoDB Anda. us-west-2

```
import { DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbClient = new DynamoDBClient({ region: 'us-west-2' });

dbClient
  .listtables(new ListTablesCommand({}))
  .then(response => {
    console.log(response.TableNames.join('\n'));
  })
  .catch((error) => {
    console.error(error);
  });
```

Koordinasikan 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.

```
const values = await Promise.all([firstPromise, secondPromise, thirdPromise]);

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

return values;
```

Browser dan dukungan Node.js untuk janji

Support untuk JavaScript janji asli (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 kompatibilitas ECMAScript](#). GitHub

Gunakan fungsi callback anonim

Setiap metode objek layanan dapat menerima fungsi callback anonim sebagai parameter terakhir. Tanda tangan fungsi callback ini adalah sebagai berikut.

```
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.

```
ec2.describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
  } else {
    console.log(data); // request succeeded
  }
});
```

Buat permintaan klien layanan

Membuat permintaan untuk AWS melayani klien sangat mudah. Versi 3 (V3) SDK untuk JavaScript memungkinkan Anda mengirim permintaan.

Note

Anda juga dapat melakukan operasi menggunakan perintah versi 2 (V2) saat menggunakan V3 SDK untuk JavaScript. Untuk informasi selengkapnya, lihat [Menggunakan perintah V2](#).

Untuk mengirim permintaan:

1. Inisialisasi objek klien dengan konfigurasi yang diinginkan, seperti AWS Region tertentu.

2. (Opsional) Buat objek permintaan JSON dengan nilai untuk permintaan, seperti nama bucket Amazon S3 tertentu. Anda dapat memeriksa parameter permintaan dengan melihat topik Referensi API untuk antarmuka dengan nama yang terkait dengan metode klien. Misalnya, jika Anda menggunakan metode *AbcCommand*klien, antarmuka permintaan adalah *AbcInput*.
3. Inisialisasi perintah layanan, opsional, dengan objek permintaan sebagai input.
4. Panggil send klien dengan objek perintah sebagai input.

Misalnya, untuk membuat daftar tabel us-west-2 Amazon DynamoDB Anda, Anda dapat melakukannya dengan `async/await`.

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

(async function() {
  const dbClient = new DynamoDBClient({ region: 'us-west-2' });
  const command = new ListTablesCommand({});

  try {
    const results = await dbClient.send(command);
    console.log(results.TableNames.join('\n'));
  } catch (err) {
    console.error(err);
  }
})();
```

Menangani tanggapan klien layanan

Setelah metode klien layanan telah dipanggil, ia mengembalikan contoh objek respon dari antarmuka dengan nama yang terkait dengan metode klien. Misalnya, jika Anda menggunakan metode *AbcCommand*klien, objek respons adalah tipe *AbcResponse*(antarmuka).

Akses data yang dikembalikan dalam respons

Objek respon berisi data, sebagai properti, yang dikembalikan oleh permintaan layanan.

Dalam [Buat permintaan klien layanan](#), `ListTablesCommand` perintah mengembalikan nama tabel di `TableNames` properti respons.

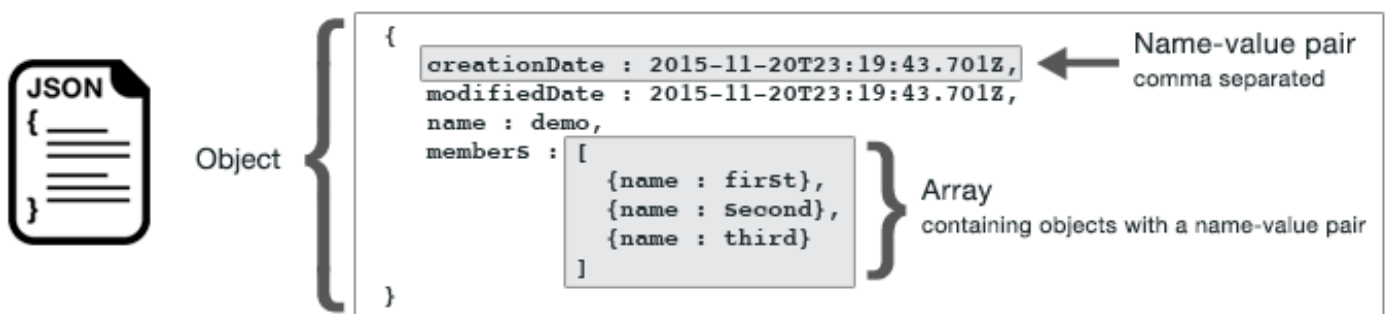
Mengakses informasi kesalahan

Jika perintah gagal, itu melempar pengecualian. Anda dapat menangani pengecualian sesuai kebutuhan.

Bekerja dengan JSON

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

AWS SDK for 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:

- Sebagai objek, yang merupakan kumpulan pasangan nama-nilai yang tidak berurutan. 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.
- Sebagai array, yang merupakan kumpulan nilai yang diurutkan. 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 AWS Lambda layanan.

```
const params = {
  FunctionName : funcName,
  Payload : JSON.stringify(payload),
  LogType : LogType.Tail,
};
```

`paramsObjek` 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 `LambdaFunctionName`, `Payload`, `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.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

SDK untuk contoh JavaScript kode

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

Temukan kode sumber untuk contoh-contoh ini dan lainnya di [Repositori Contoh AWS Kode](#) di GitHub Untuk mengusulkan contoh kode baru agar tim AWS dokumentasi mempertimbangkan untuk memproduksi, buat permintaan. 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 petunjuknya, lihat bagian Penulisan kode [di GitHub pedoman kontribusi](#).

Important

Contoh-contoh ini menggunakan sintaks impor/ekspor ECMAScript6.

- Ini memerlukan Node.js versi 14.17 atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#) pedoman konversi.

Topik

- [JavaScript ES6/CommonJS sintaks](#)
- [Contoh Amazon DynamoDB](#)
- [Contoh AWS Elemental MediaConvert](#)
- [Contoh AWS Lambda](#)
- [Contoh Amazon Lex](#)
- [Contoh Amazon Polly](#)
- [Contoh Amazon Redshift](#)
- [Contoh Layanan Email Sederhana Amazon](#)
- [Contoh Layanan Pemberitahuan Sederhana Amazon](#)
- [Contoh Amazon Transcribe](#)
- [Menyiapkan Node.js pada instans Amazon EC2](#)

- [Membangun aplikasi untuk mengirimkan data ke DynamoDB](#)
- [Membangun aplikasi transkripsi dengan pengguna yang diautentikasi](#)
- [Memanggil Lambda dengan API Gateway](#)
- [Membuat alur AWS kerja tanpa server menggunakan AWS SDK for JavaScript](#)
- [Membuat acara terjadwal untuk menjalankan AWS Lambda fungsi](#)
- [Membangun chatbot Amazon Lex](#)
- [Membuat contoh aplikasi pemesanan](#)

JavaScript ES6/CommonJS sintaks

Contoh AWS SDK for JavaScript kode ditulis dalam ECMAScript 6 (ES6). ES6 membawa sintaks baru dan fitur baru untuk membuat kode Anda lebih modern dan mudah dibaca, dan berbuat lebih banyak.

ES6 mengharuskan Anda menggunakan Node.js versi 13.x atau lebih tinggi. Untuk menginstal dan menginstal Node.js versi versi versi versi versi versi versi versi terbaru versi versi terbaru versi terbaru. [Node.js](#) Namun, jika Anda lebih suka, Anda dapat mengubah salah satu contoh kami ke sintaks CommonJS menggunakan panduan berikut:

- Hapus "type" : "module" dari `package.json` di lingkungan proyek Anda.
- Mengkonversi semua pernyataan ES6 untuk `import` pernyataan CommonJS `require`. Misalnya, konversi:

```
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "../libs/s3Client.js";
```

Untuk setara CommonJs nya:

```
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("../libs/s3Client.js");
```

- Mengkonversi semua pernyataan ES6 untuk `export` pernyataan CommonJS `module.exports`. Misalnya, konversi:

```
export {s3}
```

Untuk setara CommonJs nya:


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

Contoh berikut menunjukkan contoh kode untuk membuat bucket Amazon S3 di ES6 dan CommonJS.

ES6

libs/s3Client.js

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS region
const REGION = "eu-west-1"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
export {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using ES6 syntax.
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "../libs/s3Client.js";

// Get service clients module and commands using CommonJS syntax.
// const { CreateBucketCommand } = require("@aws-sdk/client-s3");
// const { s3 } = require("../libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

CommonJS

libs/s3Client.js

```
// Create service client module using CommonJS syntax.
const { S3Client } = require("@aws-sdk/client-s3");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
module.exports = {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using CommonJS syntax.
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("../libs/s3Client.js");

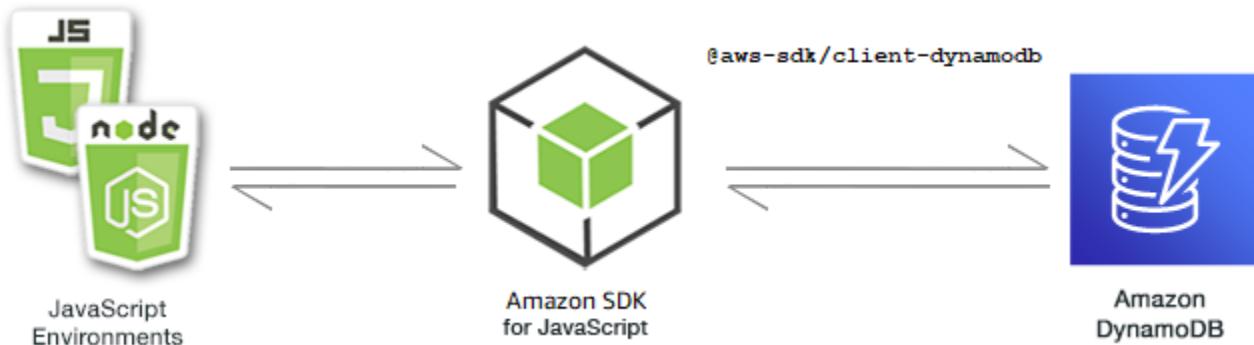
// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

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 `DynamoDBStreams`, `DynamoDB.DocumentClient` dan klien. [Untuk informasi selengkapnya tentang penggunaan kelas klien DynamoDB, lihat utilitas Class: DynamoDB, Class: DynamoDBStreams, dan Class: DynamoDB 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 DynamoDB klien berikut:

- [CreateTableCommand](#)
- [ListTablesCommand](#)
- [DescribeTableCommand](#)
- [DeleteTableCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan contoh Node.js ini, dan instal modul wajib AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Instal SDK untuk klien JavaScript DynamoDB. Untuk informasi selengkapnya, lihat [Apa yang baru di Versi 3](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh-contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduhannya](#).

Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Note

Untuk informasi tentang tipe data yang digunakan dalam contoh ini, lihat [Tipe data yang didukung dan aturan penamaan di Amazon DynamoDB](#).

Membuat tabel

Buat modul Node.js dengan nama `filecreate-table.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan `klienDynamoDB`. 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 `CreateTableCommand` metode objek layanan DynamoDB.

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

const client = new DynamoDBClient({});

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

```
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
});

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Daftar tabel Anda

Buat modul Node.js dengan nama `filelist-tables.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDBClient`. Buat objek JSON yang berisi parameter yang diperlukan untuk daftar tabel Anda, yang dalam contoh ini membatasi jumlah tabel yang terdaftar ke 10. Panggil `ListTablesCommand` metode objek layanan `DynamoDB`.

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

const client = new DynamoDBClient({});

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

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menggambarkan tabel

Buat modul Node.js dengan nama `filedescribe-table.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDB`. Buat objek JSON yang berisi parameter yang diperlukan untuk menggambarkan `DescribeTableCommand` metode objek layanan `DynamoDB`.

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

const client = new DynamoDBClient({});

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

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus tabel

Buat modul Node.js dengan nama `filedelete-table.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDB`. 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 `DeleteTableCommand` metode objek layanan `DynamoDB`.

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

const client = new DynamoDBClient({});

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

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

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

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: DynamoDB

- [PutItemCommand](#)

- [UpdateItemCommand](#)
- [GetItemCommand](#)
- [DeleteItemCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan contoh Node.js ini, dan instal modul wajib AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.
- Buat tabel DynamoDB yang itemnya dapat Anda akses. Untuk informasi selengkapnya tentang membuat tabel DynamoDB, lihat. [Membuat dan menggunakan tabel di DynamoDB](#)

Important

Contoh-contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduhannya Node.js](#).

Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Note

Untuk informasi tentang tipe data yang digunakan dalam contoh ini, lihat [Tipe data yang didukung dan aturan penamaan di Amazon DynamoDB](#).

Menulis item

Buat modul Node.js dengan nama `fileput-item.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDB`. 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 `PutItemCommand` metode objek layanan klien DynamoDB.

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

const client = new DynamoDBClient({});

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

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Perbarui item

Buat modul Node.js dengan nama `fileupdate-item.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDB`. Buat objek JSON yang berisi parameter yang diperlukan untuk menambahkan item, yang dalam contoh ini mencakup nama tabel, kunci untuk memperbarui, dan ekspresi tanggal yang memetakan nama atribut baru, dan nilai untuk setiap atribut baru. Panggil `UpdateItemCommand` metode objek layanan klien DynamoDB.

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

const client = new DynamoDBClient({});

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

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Mendapatkan item

Buat modul Node.js dengan nama `fileget-item.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDB`. Untuk mengidentifikasi item yang akan didapat, Anda harus memberikan nilai kunci utama untuk item tersebut dalam tabel. Secara default, `GetItemCommand` metode mengembalikan semua nilai atribut yang ditentukan 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 `GetItemCommand` metode objek layanan klien DynamoDB.

Contoh kode berikut mengambil item dari tabel dengan kunci primer yang hanya terdiri dari kunci partisi dan bukan dari kedua partisi dan kunci sortir. Jika tabel memiliki kunci primer yang terdiri dari kunci partisi dan kunci sortir, Anda juga harus menentukan nama kunci sortir dan atribut.

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

const client = new DynamoDBClient({});

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

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus item

Buat modul Node.js dengan nama `filedelete-item.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDB`. 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 `DeleteItemCommand` metode objek layanan klien `DynamoDB`.

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteItemCommand({
    TableName: "Drinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Name: { S: "Pumpkin Spice Latte" },
    },
  });

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node delete-item.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 dan membaca sekumpulan item. Kode menggunakan SDK for JavaScript untuk melakukan operasi baca dan tulis batch menggunakan metode kelas klien DynamoDB berikut:

- [BatchGetItemCommand](#)
- [BatchWriteItemCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.
- Buat tabel DynamoDB yang itemnya dapat Anda akses. Untuk informasi selengkapnya tentang membuat tabel DynamoDB, lihat. [Membuat dan menggunakan tabel di DynamoDB](#)

Important

Contoh-contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduhannya Node.js](#).

Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Note

Untuk informasi tentang tipe data yang digunakan dalam contoh ini, lihat [Tipe data yang didukung dan aturan penamaan di Amazon DynamoDB](#).

Membaca item dalam batch

Buat modul Node.js dengan nama `filebatch-get-item.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDBClient`. 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 `BatchGetItemCommand` metode objek layanan `DynamoDB`.

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

const client = new DynamoDBClient({});

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

  const response = await client.send(command);
```

```
console.log(response.Responses["PageAnalytics"]);
return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menulis item dalam batch

Buat modul Node.js dengan nama file `batch-write-item.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan `klienDynamoDB`. 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 `BatchWriteItemCommand` metode objek layanan DynamoDB.

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

const client = new DynamoDBClient({});

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



```
        Item: {
          Name: { S: "Donkey Kick" },
          Process: { S: "Wet-Hulled" },
          Flavors: { SS: ["Earth", "Syrup", "Spice"] },
        },
      },
    ],
  {
    PutRequest: {
      Item: {
        Name: { S: "Flora Ethiopia" },
        Process: { S: "Washed" },
        Flavors: { SS: ["Stone Fruit", "Toasted Almond", "Delicate"] },
      },
    },
  },
],
},
});

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node batch-write-item.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:

- [QueryCommand](#)
- [ScanCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan contoh Node.js ini, dan instal modul wajib AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.
- Buat tabel DynamoDB yang itemnya dapat Anda akses. Untuk informasi selengkapnya tentang membuat tabel DynamoDB, lihat. [Membuat dan menggunakan tabel di DynamoDB](#)

Important

Contoh-contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduhannya](#) [Node.js](#).

Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Note

Untuk informasi tentang tipe data yang digunakan dalam contoh ini, lihat [Tipe data yang didukung dan aturan penamaan di Amazon DynamoDB](#).

Menanyakan tabel

Buat modul Node.js dengan nama `filequery.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDBClient`. 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 `QueryCommand` metode objek layanan `DynamoDB`.

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

const client = new DynamoDBClient({});

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

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
```

```
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node query.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Memindai tabel

Buat modul Node.js dengan nama `filescan.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Untuk mengakses DynamoDB, buat objek layanan klien `DynamoDB`. 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 `ScanCommand` metode objek layanan `DynamoDB`.

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

const client = new DynamoDBClient({});

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

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
```

```
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node 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 utilitas DynamoDB.

Skenario

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

Untuk informasi selengkapnya tentang Klien Dokumen DynamoDB, [lihat @aws -sdk/lib-dynamodb README](#) di GitHub 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 utilitas DynamoDB. Kode menggunakan SDK for JavaScript untuk query dan scan tabel menggunakan metode kelas DynamoDB Document Client ini:

- [GetCommand](#)
- [PutCommand](#)
- [UpdateCommand](#)
- [QueryCommand](#)

- [DeleteCommand](#)

[Untuk informasi selengkapnya tentang mengonfigurasi Klien Dokumen DynamoDB, lihat @aws -sdk/lib-dynamodb.](#)

Tugas Prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan contoh Node.js ini, dan instal modul wajib AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.
- 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.

Important

Contoh-contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).

Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Note

Untuk informasi tentang tipe data yang digunakan dalam contoh ini, lihat [Tipe data yang didukung dan aturan penamaan di Amazon DynamoDB](#).

Mendapatkan Item dari Tabel

Buat modul Node.js dengan nama file `get.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Ini termasuk `@aws-`

`aws-sdk/lib-dynamodb`, paket pustaka yang menyediakan fungsionalitas klien dokumen untuk `aws-sdk/client-dynamodb`. Selanjutnya, atur konfigurasi seperti yang ditunjukkan di bawah ini untuk marshalling dan unmarshalling - sebagai parameter kedua opsional - selama pembuatan klien dokumen. Selanjutnya, buat klien. Sekarang 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 `GetCommand` metode Klien Dokumen DynamoDB.

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node get.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menempatkan Item di Tabel

Buat modul Node.js dengan nama file `put.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Ini termasuk `aws-sdk/lib-dynamodb`, paket pustaka yang menyediakan fungsionalitas klien dokumen untuk `aws-sdk/client-dynamodb`. Selanjutnya, atur konfigurasi seperti yang ditunjukkan di bawah ini untuk marshalling dan unmarshalling - sebagai parameter kedua opsional - selama pembuatan klien

dokumen. Selanjutnya, buat klien. Buat objek JSON yang berisi parameter yang diperlukan untuk menulis item ke tabel, yang dalam contoh ini mencakup nama tabel dan deskripsi item yang akan ditambahkan atau diperbarui yang mencakup hashkey dan nilai serta nama dan nilai untuk atribut untuk ditetapkan pada item. Panggil PutCommand metode Klien Dokumen DynamoDB.

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node put.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Memperbarui Item dalam Tabel

Buat modul Node.js dengan nama `fileupdate.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Ini termasuk `@aws-sdk/lib-dynamodb`, paket pustaka yang menyediakan fungsionalitas klien dokumen untuk `@aws-sdk/client-dynamodb`. Selanjutnya, atur konfigurasi seperti yang ditunjukkan di bawah ini untuk marshalling dan unmarshalling - sebagai parameter kedua opsional - selama pembuatan klien dokumen. Selanjutnya, buat klien. 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 yang menentukan atribut item `UpdateExpressions` yang akan diperbarui dengan token

yang Anda tetapkan nilai dalam `ExpressionAttributeValues` parameter. Panggil metode Klien Dokumen `UpdateCommand` DynamoDB.

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node update.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Melakukan Kueri Tabel

Buat modul Node.js dengan nama `filequery.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Ini termasuk `@aws-sdk/lib-dynamodb`, paket pustaka yang menyediakan fungsionalitas klien dokumen untuk `@aws-sdk/client-dynamodb`. 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 `QueryCommand` metode Klien Dokumen DynamoDB.

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node query.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus Item dari Tabel

Buat modul Node.js dengan nama `filedelete.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Ini termasuk `@aws-sdk/lib-dynamodb`, paket pustaka yang menyediakan fungsionalitas klien dokumen untuk `@aws-sdk/client-dynamodb`. Selanjutnya, atur konfigurasi seperti yang ditunjukkan di bawah ini untuk marshalling dan unmarshalling - sebagai parameter kedua opsional - selama pembuatan klien dokumen. Selanjutnya, buat klien. Untuk mengakses DynamoDB, buat objek `DynamoDB`. Buat objek JSON yang berisi parameter yang diperlukan untuk menghapus item dalam tabel, yang dalam

contoh ini mencakup nama tabel dan nama serta nilai hashkey item yang ingin Anda hapus. Panggil `DeleteCommand` metode Klien Dokumen DynamoDB.

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node delete.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Contoh AWS Elemental MediaConvert

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 `MediaConvert` klien. Untuk informasi selengkapnya, lihat [Kelas: MediaConvert](#) di Referensi API.

Topik

- [Mendapatkan titik akhir khusus wilayah Anda untuk MediaConvert](#)

- [Membuat dan mengelola pekerjaan transcoding di MediaConvert](#)
- [Menggunakan template pekerjaan 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

- [DescribeEndpointsCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.
- 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 MediaConvertPanduan Pengguna.

⚠ Important

Contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduhannya Node.js](#). Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Mendapatkan URL endpoint Anda

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileemcClientGet.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang menciptakan objek MediaConvert klien. Ganti **REGION** dengan AWS Region Anda.

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileemc_getendpoint.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk melewati parameter permintaan kosong untuk `DescribeEndpointsCommand` metode kelas MediaConvert klien. Kemudian panggil `DescribeEndpointsCommand` metodenya.

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

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

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

```
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
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:

- [CreateJobCommand](#)
- [CancelJobCommand](#)
- [GetJobCommand](#)
- [ListJobsCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.
- Buat dan konfigurasi 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 MediaConvertPanduan Pengguna.

Important

Contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#). Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Mengkonfigurasi SDK

Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Karena MediaConvert menggunakan titik akhir khusus untuk setiap akun, Anda juga harus mengonfigurasi kelas MediaConvert klien untuk menggunakan titik akhir khusus wilayah Anda. Untuk melakukan ini, atur endpoint parameter pada `mediaconvert(endpoint)`.

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

Mendefinisikan pekerjaan transcoding sederhana

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileemcClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang menciptakan objek `MediaConvert` klien. Ganti **REGION** dengan AWS Region Anda. Ganti **ENDPOINT** dengan titik akhir `MediaConvert` akun Anda, yang dapat Anda lakukan di halaman Akun di konsol. `MediaConvert`

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileemc_createjob.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Buat JSON yang mendefinisikan parameter pekerjaan transcode.

Parameter ini cukup rinci. Anda dapat menggunakan [AWS Elemental MediaConvertkonsol](#) 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.

Note

Ganti `JOB_QUEUE_ARN` dengan antrian `MediaConvert` pekerjaan, `IAM_ROLE_ARN` dengan Amazon Resource Name (ARN) dari peran IAM, `OUTPUT_BUCKET_NAME` dengan nama bucket tujuan - misalnya, `"s3://OUTPUT_BUCKET_NAME/"`, dan `INPUT_BUCKET_AND_FILENAME` dengan bucket input dan nama file - misalnya, `"s3://INPUT_BUCKET/FILE_NAME"`.

```
const params = {
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
```



```
UserMetadata: {
  Customer: "Amazon",
},
Role: "IAM_ROLE_ARN", //IAM_ROLE_ARN
Settings: {
  OutputGroups: [
    {
      Name: "File Group",
      OutputGroupSettings: {
        Type: "FILE_GROUP_SETTINGS",
        FileGroupSettings: {
          Destination: "OUTPUT_BUCKET_NAME", //OUTPUT_BUCKET_NAME, e.g., "s3://
BUCKET_NAME/"
        },
      },
    },
  ],
  Outputs: [
    {
      VideoDescription: {
        ScalingBehavior: "DEFAULT",
        TimecodeInsertion: "DISABLED",
        AntiAlias: "ENABLED",
        Sharpness: 50,
        CodecSettings: {
          Codec: "H_264",
          H264Settings: {
            InterlaceMode: "PROGRESSIVE",
            NumberReferenceFrames: 3,
            Syntax: "DEFAULT",
            Softness: 0,
            GopClosedCadence: 1,
            GopSize: 90,
            Slices: 1,
            GopBReference: "DISABLED",
            SlowPal: "DISABLED",
            SpatialAdaptiveQuantization: "ENABLED",
            TemporalAdaptiveQuantization: "ENABLED",
            FlickerAdaptiveQuantization: "DISABLED",
            EntropyEncoding: "CABAC",
            Bitrate: 5000000,
            FramerateControl: "SPECIFIED",
            RateControlMode: "CBR",
            CodecProfile: "MAIN",
            Telecine: "NONE",
            MinIInterval: 0,
```

```
        AdaptiveQuantization: "HIGH",
        CodecLevel: "AUTO",
        FieldEncoding: "PAFF",
        SceneChangeDetect: "ENABLED",
        QualityTuningLevel: "SINGLE_PASS",
        FramerateConversionAlgorithm: "DUPLICATE_DROP",
        UnregisteredSeiTimecode: "DISABLED",
        GopSizeUnits: "FRAMES",
        ParControl: "SPECIFIED",
        NumberBFramesBetweenReferenceFrames: 2,
        RepeatPps: "DISABLED",
        FramerateNumerator: 30,
        FramerateDenominator: 1,
        ParNumerator: 1,
        ParDenominator: 1,
    },
},
AfdSignaling: "NONE",
DropFrameTimecode: "ENABLED",
RespondToAfd: "NONE",
ColorMetadata: "INSERT",
},
AudioDescriptions: [
    {
        AudioTypeControl: "FOLLOW_INPUT",
        CodecSettings: {
            Codec: "AAC",
            AacSettings: {
                AudioDescriptionBroadcasterMix: "NORMAL",
                RateControlMode: "CBR",
                CodecProfile: "LC",
                CodingMode: "CODING_MODE_2_0",
                RawFormat: "NONE",
                SampleRate: 48000,
                Specification: "MPEG4",
                Bitrate: 64000,
            },
        },
        LanguageCodeControl: "FOLLOW_INPUT",
        AudioSourceName: "Audio Selector 1",
    },
],
ContainerSettings: {
    Container: "MP4",
```

```
        Mp4Settings: {
            CslgAtom: "INCLUDE",
            FreeSpaceBox: "EXCLUDE",
            MoovPlacement: "PROGRESSIVE_DOWNLOAD",
        },
    },
    NameModifier: "_1",
},
],
},
AdAvailOffset: 0,
Inputs: [
    {
        AudioSelectors: {
            "Audio Selector 1": {
                Offset: 0,
                DefaultSelection: "NOT_DEFAULT",
                ProgramSelection: 1,
                SelectorType: "TRACK",
                Tracks: [1],
            },
        },
        VideoSelector: {
            ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
"s3://BUCKET_NAME/FILE_NAME"
    },
],
    TimecodeConfig: {
        Source: "EMBEDDED",
    },
},
};
```

Membuat pekerjaan transcoding

Setelah membuat parameter pekerjaan JSON, panggil `run` metode asinkron untuk memanggil objek layanan MediaConvert klien, melewati parameter. ID pekerjaan yang dibuat dikembalikan dalam `responsdata`.

```
const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Job created!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node emc_createjob.js
```

Kode contoh lengkap ini dapat ditemukan [di sini GitHub](#).

Membatalkan pekerjaan transcoding

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileemcClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang menciptakan objek MediaConvert klien. Ganti **REGION** dengan AWS Region Anda. Ganti **ENDPOINT** dengan titik akhir MediaConvert akun Anda, yang dapat Anda lakukan di halaman Akun di konsol. MediaConvert

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileemc_canceljob.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. Buat JSON yang menyertakan ID pekerjaan yang akan dibatalkan. Kemudian panggil `CancelJobCommand` metode dengan membuat janji untuk memanggil objek layanan `MediaConvert` klien, melewati parameter. Menangani respon dalam callback janji.

Note

Ganti `JOB_ID` dengan ID pekerjaan yang akan dibatalkan.

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

// Set the parameters
const params = { Id: "JOB_ID" }; //JOB_ID

const run = async () => {
  try {
    const data = await emcClient.send(new CancelJobCommand(params));
    console.log("Job " + params.Id + " is canceled");
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node ec2_canceljob.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Daftar pekerjaan transcoding terbaru

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileemcClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang menciptakan objek `MediaConvert` klien. Ganti `REGION` dengan AWS Region Anda. Ganti `ENDPOINT` dengan titik akhir `MediaConvert` akun Anda, yang dapat Anda lakukan di halaman Akun di konsol. `MediaConvert`

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileemc_listjobs.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat parameter JSON, termasuk nilai untuk menentukan apakah akan mengurutkan daftar dalam ASCENDING, atau DESCENDING urutan, Amazon Resource Name (ARN) dari antrian pekerjaan yang akan diperiksa, dan status pekerjaan yang akan disertakan. Kemudian panggil `ListJobsCommand` metode dengan membuat janji untuk memanggil objek layanan MediaConvert klien, melewati parameter.

Note

Ganti `QUEUE_ARN` dengan Amazon Resource Name (ARN) dari antrian pekerjaan yang akan diperiksa, dan `STATUS` dengan status antrian.

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

// Set the parameters
const params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED", // e.g., "SUBMITTED"
};

const run = async () => {
  try {
```

```
const data = await emcClient.send(new ListJobsCommand(params));
console.log("Success. Jobs: ", data.Jobs);
} catch (err) {
  console.log("Error", err);
}
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node emc_listjobs.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menggunakan template pekerjaan di MediaConvert



Contoh kode Node.js ini menunjukkan:

- Cara membuat template AWS Elemental 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:

- [CreateJobTemplateCommand](#)

- [CreateJobCommand](#)
- [DeleteJobTemplateCommand](#)
- [ListJobTemplatesCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.
- 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 MediaConvertPanduan Pengguna.

Important

Contoh-contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).

Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Membuat template pekerjaan

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileemcClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang menciptakan objek MediaConvert klien. Ganti **REGION** dengan AWS Region Anda. Ganti **ENDPOINT** dengan titik akhir MediaConvert akun Anda, yang dapat Anda lakukan di halaman Akun di konsol. MediaConvert

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



```
};  
// Set the MediaConvert Service Object  
const emcClient = new MediaConvertClient(ENDPOINT);  
export { emcClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileemc_create_jobtemplate.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

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 dari [Membuat dan mengelola pekerjaan transcoding di MediaConvert](#).

Panggil `CreateJobTemplateCommand` metode dengan membuat janji untuk memanggil objek layanan MediaConvert klien, melewati parameter.

Note

Ganti `JOB_QUEUE_ARN` dengan Amazon Resource Name (ARN) dari antrian pekerjaan yang akan diperiksa, dan `BUCKET_NAME` dengan nama bucket Amazon S3 tujuan - misalnya, `"s3://BUCKET_NAME/"`.

```
// Import required AWS-SDK clients and commands for Node.js  
import { CreateJobTemplateCommand } from "@aws-sdk/client-mediaconvert";  
import { emcClient } from "../libs/emcClient.js";  
  
const params = {  
  Category: "YouTube Jobs",  
  Description: "Final production transcode",  
  Name: "DemoTemplate",  
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN  
  Settings: {  
    OutputGroups: [  
      {  
        Name: "File Group",  
        OutputGroupSettings: {  
          Type: "FILE_GROUP_SETTINGS",
```

```
FileGroupSettings: {
  Destination: "BUCKET_NAME", // BUCKET_NAME e.g., "s3://BUCKET_NAME/"
},
},
Outputs: [
{
  VideoDescription: {
    ScalingBehavior: "DEFAULT",
    TimecodeInsertion: "DISABLED",
    AntiAlias: "ENABLED",
    Sharpness: 50,
    CodecSettings: {
      Codec: "H_264",
      H264Settings: {
        InterlaceMode: "PROGRESSIVE",
        NumberReferenceFrames: 3,
        Syntax: "DEFAULT",
        Softness: 0,
        GopClosedCadence: 1,
        GopSize: 90,
        Slices: 1,
        GopBReference: "DISABLED",
        SlowPal: "DISABLED",
        SpatialAdaptiveQuantization: "ENABLED",
        TemporalAdaptiveQuantization: "ENABLED",
        FlickerAdaptiveQuantization: "DISABLED",
        EntropyEncoding: "CABAC",
        Bitrate: 5000000,
        FramerateControl: "SPECIFIED",
        RateControlMode: "CBR",
        CodecProfile: "MAIN",
        Telecine: "NONE",
        MinIInterval: 0,
        AdaptiveQuantization: "HIGH",
        CodecLevel: "AUTO",
        FieldEncoding: "PAFF",
        SceneChangeDetect: "ENABLED",
        QualityTuningLevel: "SINGLE_PASS",
        FramerateConversionAlgorithm: "DUPLICATE_DROP",
        UnregisteredSeiTimecode: "DISABLED",
        GopSizeUnits: "FRAMES",
        ParControl: "SPECIFIED",
        NumberBFramesBetweenReferenceFrames: 2,
        RepeatPps: "DISABLED",
```

```

        FramerateNumerator: 30,
        FramerateDenominator: 1,
        ParNumerator: 1,
        ParDenominator: 1,
    },
},
AfdSignaling: "NONE",
DropFrameTimecode: "ENABLED",
RespondToAfd: "NONE",
ColorMetadata: "INSERT",
},
AudioDescriptions: [
    {
        AudioTypeControl: "FOLLOW_INPUT",
        CodecSettings: {
            Codec: "AAC",
            AacSettings: {
                AudioDescriptionBroadcasterMix: "NORMAL",
                RateControlMode: "CBR",
                CodecProfile: "LC",
                CodingMode: "CODING_MODE_2_0",
                RawFormat: "NONE",
                SampleRate: 48000,
                Specification: "MPEG4",
                Bitrate: 64000,
            },
        },
        LanguageCodeControl: "FOLLOW_INPUT",
        AudioSourceName: "Audio Selector 1",
    },
],
ContainerSettings: {
    Container: "MP4",
    Mp4Settings: {
        CslgAtom: "INCLUDE",
        FreeSpaceBox: "EXCLUDE",
        MoovPlacement: "PROGRESSIVE_DOWNLOAD",
    },
},
NameModifier: "_1",
},
],
},
],

```

```
AdAvailOffset: 0,
Inputs: [
  {
    AudioSelectors: {
      "Audio Selector 1": {
        Offset: 0,
        DefaultSelection: "NOT_DEFAULT",
        ProgramSelection: 1,
        SelectorType: "TRACK",
        Tracks: [1],
      },
    },
    VideoSelector: {
      ColorSpace: "FOLLOW",
    },
    FilterEnable: "AUTO",
    PsiControl: "USE_PSI",
    FilterStrength: 0,
    DeblockFilter: "DISABLED",
    DenoiseFilter: "DISABLED",
    TimecodeSource: "EMBEDDED",
  },
],
TimecodeConfig: {
  Source: "EMBEDDED",
},
},
];

const run = async () => {
  try {
    // Create a promise on a MediaConvert object
    const data = await emcClient.send(new CreateJobTemplateCommand(params));
    console.log("Success!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node emc_create_jobtemplate.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Membuat pekerjaan transcoding dari template pekerjaan

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileemcClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang menciptakan objek MediaConvert klien. Ganti **REGION** dengan AWS Region Anda. Ganti **ENDPOINT** dengan titik akhir MediaConvert akun Anda, yang dapat Anda lakukan di halaman Akun di konsol. MediaConvert

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileemc_template_createjob.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

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

Note

Ganti `JOB_QUEUE_ARN` dengan Amazon Resource Name (ARN) dari antrian pekerjaan untuk memeriksa, `KEY_PAIR_NAME` dengan, `TEMPLATE_NAME` dengan, `ROLE_ARN` dengan Amazon Resource Name (ARN) peran, dan `INPUT_BUCKET_AND_FILENAME` dengan bucket input dan nama file - misalnya, `"s3://BUCKET_NAME/FILE_NAME"`.

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

const params = {
  Queue: "QUEUE_ARN", //QUEUE_ARN
  JobTemplate: "TEMPLATE_NAME", //TEMPLATE_NAME
  Role: "ROLE_ARN", //ROLE_ARN
  Settings: {
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
        "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
  },
};

const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Success! ", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
}  
};  
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node emc_template_createjob.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Daftar template pekerjaan Anda

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileemcClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang menciptakan objek MediaConvert klien. Ganti **REGION** dengan AWS Region Anda. Ganti **ENDPOINT** dengan titik akhir MediaConvert akun Anda, yang dapat Anda lakukan di halaman Akun di konsol. MediaConvert

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileemc_listtemplates.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk melewati parameter permintaan untuk `listTemplates` metode kelas 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 `ListTemplatesCommand` metode, buat janji untuk memanggil objek layanan MediaConvert klien, melewati parameter.

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

```
import { emcClient } from "./libs/emcClient.js";

const params = {
  ListBy: "NAME",
  MaxResults: 10,
  Order: "ASCENDING",
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobTemplatesCommand(params));
    console.log("Success ", data.JobTemplates);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node emc_listtemplates.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus template pekerjaan

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileemcClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang menciptakan objek MediaConvert klien. Ganti **REGION** dengan AWS Region Anda. Ganti **ENDPOINT** dengan titik akhir MediaConvert akun Anda, yang dapat Anda lakukan di halaman Akun di konsol. MediaConvert

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileemc_deletetemplate.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk meneruskan nama template pekerjaan yang ingin Anda hapus sebagai parameter untuk `DeleteJobTemplateCommand` metode kelas `MediaConvert` klien. Untuk memanggil `DeleteJobTemplateCommand` metode, buat janji untuk memanggil objek layanan `MediaConvert` klien, melewati parameter.

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

// Set the parameters
const params = { Name: "test" }; //TEMPLATE_NAME

const run = async () => {
  try {
    const data = await emcClient.send(new DeleteJobTemplateCommand(params));
    console.log(
      "Success, template deleted! Request ID:",
      data.$metadata.requestId,
    );
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node emc_deletetemplate.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Contoh AWS Lambda

AWS Lambda adalah layanan komputasi tanpa server yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server, membuat logika penskalaan kluster yang sadar beban kerja, mempertahankan integrasi peristiwa, atau mengelola waktu kerja.

JavaScript API untuk AWS Lambda diekspos melalui kelas [LambdaService](#) klien.

Berikut adalah daftar contoh yang menunjukkan cara membuat dan menggunakan fungsi Lambda dengan v3: AWS SDK for JavaScript

- [Memanggil Lambda dengan API Gateway](#)
- [Membuat acara terjadwal untuk menjalankan AWS Lambda fungsi](#)

Contoh Amazon Lex

Amazon Lex adalah AWS layanan untuk membangun antarmuka percakapan ke dalam aplikasi menggunakan suara dan teks.

JavaScriptAPI untuk Amazon Lex diekspos melalui kelas klien [Lex Runtime Service](#).

- [Membangun chatbot Amazon Lex](#)

Contoh Amazon Polly



Contoh kode Node.js ini menunjukkan:

- Unggah audio yang direkam menggunakan Amazon Polly ke Amazon S3

Skenario

Dalam contoh ini, serangkaian modul Node.js digunakan untuk mengunggah audio yang direkam secara otomatis menggunakan Amazon Polly ke Amazon S3 menggunakan metode kelas klien Amazon S3 ini:

- [StartSpeechSynthesisTaskCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan JavaScript contoh Node dengan mengikuti petunjuk pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.
- Buat AWS Identity and Access Management (IAM) Peran pengguna Amazon Cognito yang tidak diautentikasi polly: izinSynthesizeSpeech , dan kumpulan identitas Amazon Cognito dengan peran IAM yang dilampirkan padanya. [Buat AWS sumber daya menggunakan AWS CloudFormation](#)Bagian di bawah ini menjelaskan cara membuat sumber daya ini.

Note

Contoh ini menggunakan Amazon Cognito, tetapi jika Anda tidak menggunakan Amazon Cognito maka pengguna AWS Anda harus mengikuti kebijakan izin IAM

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "mobileanalytics:PutEvents",
        "cognito-sync:*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "polly:SynthesizeSpeech",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Buat AWS sumber daya menggunakan AWS CloudFormation

AWS CloudFormation memungkinkan Anda untuk membuat dan menyediakan penyebaran AWS infrastruktur yang dapat diprediksi dan berulang kali. Untuk informasi selengkapnya tentang AWS CloudFormation, lihat [AWS CloudFormation Panduan Pengguna](#).

Untuk membuat AWS CloudFormation tumpukan:

1. Instal dan konfigurasi petunjuk AWS CLI berikut di [Panduan AWS CLI Pengguna](#).
2. Buat file bernama `setup.yaml` di direktori root folder proyek Anda, dan salin konten [di sini GitHub](#) ke dalamnya.

Note

AWS CloudFormationTemplate dibuat menggunakan yang AWS CDK tersedia [di sini GitHub](#). Untuk informasi selengkapnya tentang AWS CDK, lihat [Panduan Developer AWS Cloud Development Kit \(AWS CDK\)](#).

3. Jalankan perintah berikut dari baris perintah, ganti `STACK_NAME` dengan *nama* unik untuk tumpukan.

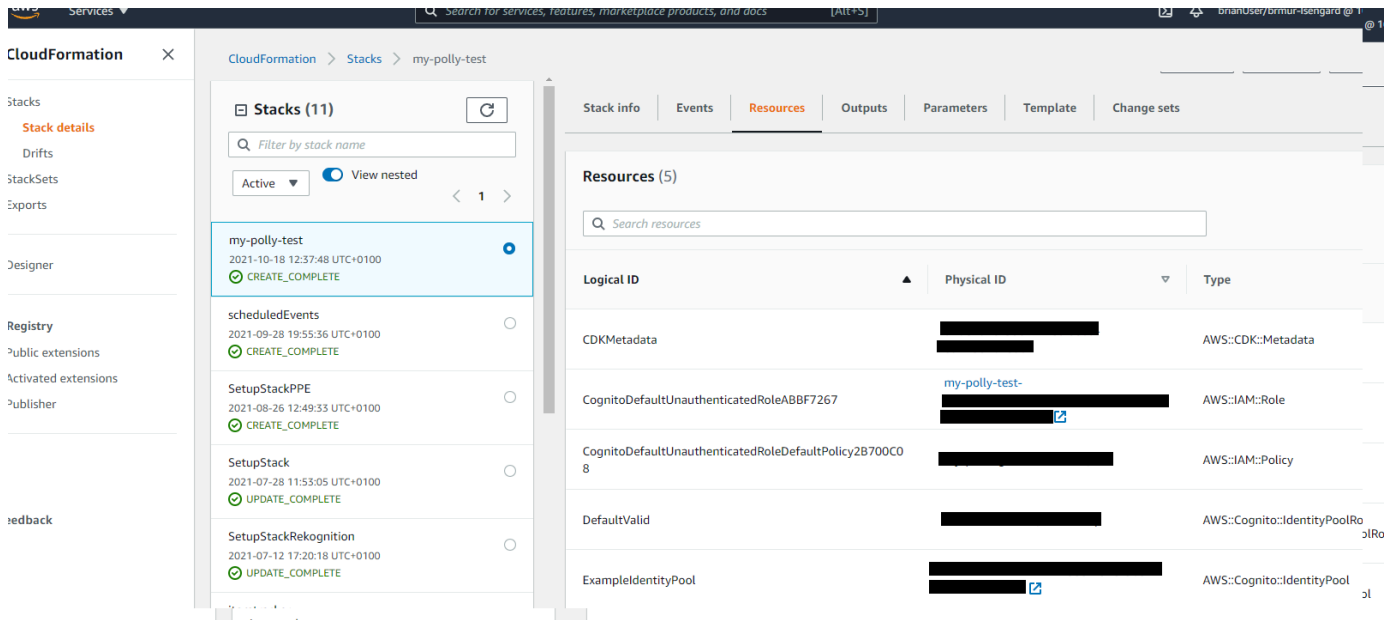
Important

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Untuk informasi selengkapnya tentang parameter `create-stack` perintah, lihat [panduan Referensi AWS CLI Perintah](#), dan [Panduan AWS CloudFormation Pengguna](#).

4. Arahkan ke konsol AWS CloudFormation manajemen, pilih Tumpukan, pilih nama tumpukan, dan pilih tab Sumber Daya untuk melihat daftar sumber daya yang dibuat.



Unggah audio yang direkam menggunakan Amazon Polly ke Amazon S3

Buat modul Node.js dengan nama `filepolly_synthesize_to_s3.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Dalam kode, masukkan **REGION**, dan **BUCKET_NAME**. Untuk mengakses Amazon Polly, buat objek layanan Polly klien. Ganti **"IDENTITY_POOL_ID"** dengan *halaman IdentityPoolId* dari Sample dari kumpulan identitas Amazon Cognito yang Anda buat untuk contoh ini. Ini juga diteruskan ke setiap objek klien.

Panggil `StartSpeechSynthesisCommand` metode objek layanan klien Amazon Polly mensintesis pesan suara dan mengunggahnya ke bucket Amazon S3.

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

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

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

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Contoh Amazon Redshift

Amazon Redshift adalah layanan gudang data dengan skala petabyte yang terkelola penuh di cloud. Gudang data Amazon Redshift adalah kumpulan sumber daya komputasi yang disebut node, yang diatur ke dalam grup yang disebut cluster. Setiap kluster menjalankan mesin Amazon Redshift dan berisi satu atau lebih database.



JavaScript API untuk Amazon Redshift diekspos melalui kelas klien [Amazon Redshift](#).

Topik

- [Contoh Amazon Redshift](#)

Contoh Amazon Redshift

Dalam contoh ini, serangkaian modul Node.js digunakan untuk membuat, memodifikasi, menjelaskan parameter, dan kemudian menghapus kluster Amazon Redshift menggunakan metode berikut dari kelas klien: `Redshift`

- [CreateClusterCommand](#)
- [ModifyClusterCommand](#)
- [DescribeClustersCommand](#)
- [DeleteClusterCommand](#)

Untuk informasi selengkapnya tentang pengguna Amazon Redshift, lihat panduan memulai [Amazon Redshift](#).

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#)

Membuat cluster Amazon Redshift

Contoh ini menunjukkan cara membuat cluster Amazon Redshift menggunakan file. AWS SDK for JavaScript Untuk informasi lebih lanjut, lihat [CreateCluster](#).

⚠ Important

Cluster yang akan Anda buat adalah live (dan tidak berjalan di kotak pasir). Anda dikenakan biaya penggunaan Amazon Redshift standar untuk kluster hingga Anda menghapusnya. Jika Anda menghapus cluster di tempat yang sama seperti saat Anda membuatnya, total biaya minimal.

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileRedshiftClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Redshift. Ganti **REGION** dengan AWS Region Anda.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileRedshift-create-cluster.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Buat objek parameter, tentukan tipe node yang akan disediakan, dan kredensial masuk master untuk instance database yang dibuat secara otomatis di cluster, dan terakhir tipe cluster.

📘 Note

Ganti **CLUSTER_NAME** dengan nama cluster. Untuk **NODE_TYPE** tentukan tipe node yang akan disediakan, seperti `'dc2.large'`, misalnya. **MASTER_USERNAME** dan **MASTER_USER_PASSWORD** adalah kredensial masuk dari pengguna utama instans DB Anda di cluster. Untuk **CLUSTER_TYPE**, masukkan tipe cluster. Jika Anda menentukansingle-node, Anda tidak memerlukan `NumberOfNodes` parameter. Parameter yang tersisa adalah opsional.

```
// Import required AWS SDK clients and commands for Node.js
```



```
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least one
  uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
  your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
  cluster subnet group to be associated with this cluster. Defaults to 'default' if not
  specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node redshift-create-cluster.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Memodifikasi cluster Amazon Redshift

Contoh ini menunjukkan cara memodifikasi kata sandi pengguna utama cluster Amazon Redshift menggunakan file. AWS SDK for JavaScript Untuk informasi selengkapnya tentang pengaturan lain yang dapat Anda ubah, lihat [ModifyCluster](#).

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileredshiftClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Redshift. Ganti **REGION** dengan AWS Region Anda.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileredshift-modify-cluster.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Tentukan AWS Wilayah, nama cluster yang ingin Anda modifikasi, dan kata sandi pengguna master baru.

Note

Ganti **CLUSTER_NAME** dengan *nama* cluster, dan **MASTER_USER_PASSWORD** dengan *kata sandi pengguna master* baru.

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

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
}  
};  
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node redshift-modify-cluster.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Melihat detail cluster Amazon Redshift

Contoh ini menunjukkan cara melihat detail cluster Amazon Redshift menggunakan file. AWS SDK for JavaScript Untuk informasi lebih lanjut tentang opsional, lihat [DescribeClusters](#).

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileredshiftClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Redshift. Ganti **REGION** dengan AWS Region Anda.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create Redshift service object.  
const redshiftClient = new RedshiftClient({ region: REGION });  
export { redshiftClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileredshift-describe-clusters.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Tentukan AWS Wilayah, nama cluster yang ingin Anda modifikasi, dan kata sandi pengguna master baru.

Note

Ganti **CLUSTER_NAME** dengan nama cluster.

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

```
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node redshift-describe-clusters.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus klaster Amazon Redshift

Contoh ini menunjukkan cara melihat detail cluster Amazon Redshift menggunakan file. AWS SDK for JavaScript Untuk informasi selengkapnya tentang pengaturan lain yang dapat Anda ubah, lihat [DeleteCluster](#).

Buat `libs` direktori, dan buat modul Node.js dengan nama `fileredshiftClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Redshift. Ganti **REGION** dengan AWS Region Anda.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan file bernama `redshift-delete-clusters.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Tentukan AWS Wilayah, nama cluster yang ingin Anda modifikasi, dan kata sandi pengguna master baru. Tentukan apakah Anda ingin menyimpan snapshot akhir cluster sebelum menghapus, dan jika demikian ID snapshot.

Note

Ganti `CLUSTER_NAME` dengan nama cluster. Untuk `SkipFinalClusterSnapshot`, tentukan apakah akan membuat snapshot akhir cluster sebelum menghapusnya. *Jika Anda menentukan 'false', tentukan id snapshot cluster terakhir di `CLUSTER_SNAPSHOT_ID`.* Anda bisa mendapatkan ID ini dengan mengklik tautan di kolom Snapshots untuk cluster di dasbor Clusters, dan menggulir ke bawah ke panel Snapshots. Perhatikan bahwa batang `rs`: bukan bagian dari ID snapshot.

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

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

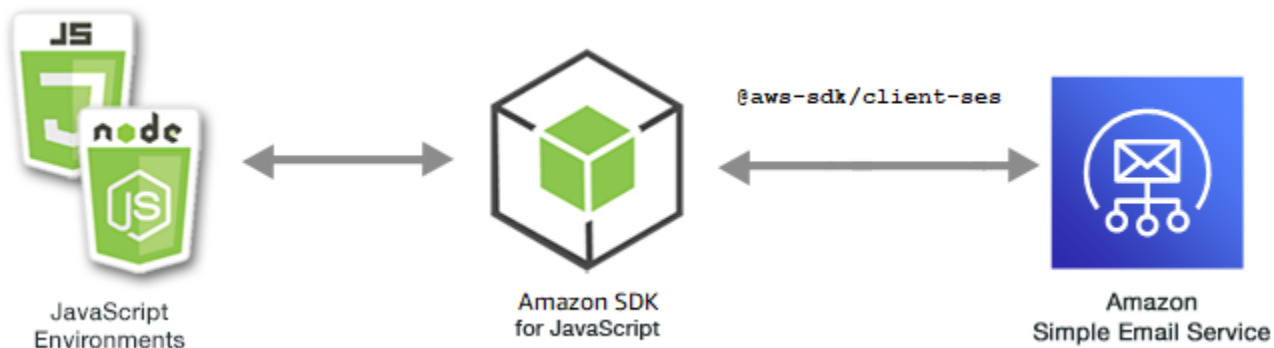
Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node redshift-delete-cluster.js
```

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 SES klien. Untuk informasi selengkapnya tentang menggunakan kelas klien Amazon SES, lihat [Kelas: SES](#) di Referensi API.

Topik

- [Mengelola identitas Amazon SES](#)
- [Bekerja dengan template email di Amazon SES](#)
- [Mengirim email menggunakan 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 AWS Identity and Access Management (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 SES klien berikut:

- [ListIdentitiesCommand](#)
- [DeleteIdentityCommand](#)
- [VerifyEmailIdentityCommand](#)
- [VerifyDomainIdentityCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWS SDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduhannya Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#)

Daftar identitas Anda

Dalam contoh ini, gunakan modul Node.js untuk mencantumkan alamat email dan domain yang akan digunakan dengan Amazon SES.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_listidentities.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk melewati `IdentityType` dan parameter lainnya untuk `ListIdentitiesCommand` metode kelas SES klien. Untuk memanggil `ListIdentitiesCommand` metode, panggil objek layanan Amazon SES, melewati objek parameter.

Yang data dikembalikan berisi array identitas domain seperti yang ditentukan oleh `IdentityType` parameter.

Note

Ganti **IDENTITY_TYPE** dengan tipe identitas, yang dapat berupa "EmailAddress" atau "Domain".


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

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
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 `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_verifyemailidentity.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan.

Buat objek untuk melewati `EmailAddress` parameter untuk `VerifyEmailIdentityCommand` metode kelas SES klien. Untuk memanggil `VerifyEmailIdentityCommand` metode, panggil objek layanan klien Amazon SES, meneruskan parameter.

Note

Ganti `ADDRESS@DOMAIN.EXT` dengan alamat email, seperti `name@example.com`.

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

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. 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 `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_verifydomainidentity.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk melewati `Domain` parameter untuk `VerifyDomainIdentityCommand` metode kelas SES klien. Untuk memanggil `VerifyDomainIdentityCommand` metode, panggil objek layanan klien Amazon SES, meneruskan objek parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan `send` metode dalam pola `async/await`. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti **AMI_ID** dengan ID Amazon Machine Image (AMI) untuk dijalankan, dan **KEY_PAIR_NAME** dari key pair yang akan ditetapkan ke ID AMI.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
```

```
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. 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 `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_deleteidentity.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk melewati Identity parameter untuk DeleteIdentityCommand metode kelas SES klien. Untuk memanggil DeleteIdentityCommand metode, buat request untuk memanggil objek layanan klien Amazon SES, melewati parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan send metode dalam pola async/await. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti *IDENTITY_TYPE* dengan tipe identitas yang akan dihapus, dan *IDENTITY_NAME* dengan nama identitas yang akan dihapus.

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

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};
```

```
const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
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:

- Cara mendapatkan daftar semua template email Anda.
- Cara mengambil dan memperbarui template email.
- Cara membuat dan menghapus 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 SES, 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 SES klien berikut:

- [ListTemplatesCommand](#)
- [CreateTemplateCommand](#)
- [GetTemplateCommand](#)
- [DeleteTemplateCommand](#)
- [UpdateTemplateCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWS SDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#)

Daftar template email Anda

Dalam contoh ini, gunakan modul Node.js untuk membuat template email untuk digunakan dengan Amazon SES.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_listtemplates.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk melewati parameter untuk `ListTemplatesCommand` metode kelas SES klien. Untuk memanggil `ListTemplatesCommand` metode, panggil objek layanan klien Amazon SES, meneruskan parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan `send` metode dalam pola `async/await`. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti *ITEMS_COUNT* dengan *jumlah* maksimum template untuk kembali. Nilai harus minimal 1 dan maksimal 10.

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

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);
```



```
try {
  return await sesClient.send(listTemplatesCommand);
} catch (err) {
  console.log("Failed to list templates.", err);
  return err;
}
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. 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 `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_gettemplate.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk melewati `TemplateName` parameter untuk `GetTemplateCommand` metode kelas SES klien. Untuk memanggil `GetTemplateCommand` metode, panggil objek layanan klien Amazon SES, meneruskan parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan send metode dalam pola async/await. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti `TEMPLATE_NAME` dengan nama template yang akan dikembalikan.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (err) {
    console.log("Failed to get email template.", err);
    return err;
  }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. 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 `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_createtemplate.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk melewati parameter untuk `CreateTemplateCommand` metode kelas SES klien, termasuk, `TemplateName`, `HtmlPartSubjectPart`, dan `TextPart`. Untuk memanggil `CreateTemplateCommand` metode, panggil objek layanan klien Amazon SES, meneruskan parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan `send` metode dalam pola `async/await`. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan `send` metode dalam pola `async/await`. Anda dapat membuat

contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti *TEMPLATE_NAME* dengan nama untuk template baru, *HTML_CONTENT* dengan *HTML tag konten* email, *SUBJECT* dengan *subjek* email, dan *TEXT_CONTENT* dengan *teks* email.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
          Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();
```

```
try {
  return await sesClient.send(createTemplateCommand);
} catch (err) {
  console.log("Failed to create template.", err);
  return err;
}
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. Template ditambahkan ke Amazon SES.

```
node ses_createtemplate.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Memperbarui templat email

Dalam contoh ini, gunakan modul Node.js untuk membuat template email untuk digunakan dengan Amazon SES.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_updatetemplate.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk meneruskan nilai `Template` parameter yang ingin Anda perbarui di template, dengan `TemplateName` parameter yang diperlukan diteruskan ke `UpdateTemplateCommand` metode kelas SES klien. Untuk memanggil `UpdateTemplateCommand` metode, panggil objek layanan Amazon SES, meneruskan parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan send metode dalam pola async/await. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti *TEMPLATE_NAME* dengan nama template, *HTML_CONTENT* dengan *HTML tag konten* email, *SUBJECT* dengan *subjek* email, dan *TEXT_CONTENT* dengan *teks* email.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

```
}  
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. Amazon SES mengembalikan detail template.

```
node ses_updatetemplate.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus templat email

Dalam contoh ini, gunakan modul Node.js untuk membuat template email untuk digunakan dengan Amazon SES.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";  
// Set the AWS Region.  
const REGION = "us-east-1";  
// Create SES service object.  
const sesClient = new SESClient({ region: REGION });  
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).


Buat modul Node.js dengan nama `fileses_deletetemplate.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk meneruskan `TemplateName` parameter yang diperlukan ke `DeleteTemplateCommand` metode kelas SES klien. Untuk memanggil `DeleteTemplateCommand` metode, panggil objek layanan Amazon SES, meneruskan parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan `send` metode dalam pola `async/await`. Anda dapat membuat

contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

 Note

Ganti *TEMPLATE_NAME* dengan nama template yang akan dihapus.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. 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 SES klien berikut:

- [SendEmailCommand](#)
- [SendTemplatedEmailCommand](#)
- [SendBulkTemplatedEmailCommand](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWS SDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#)

Persyaratan pengiriman pesan email

Amazon SES membuat pesan email dan segera mengantri untuk dikirim. Untuk mengirim email menggunakan `SendEmailCommand` 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 (yaitu, 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 `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_sendemail.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk meneruskan nilai parameter yang menentukan email yang akan dikirim, termasuk alamat pengirim dan penerima, subjek, dan badan email dalam format teks biasa dan HTML, ke `SendEmailCommand` metode kelas SES klien. Untuk memanggil `SendEmailCommand` metode, panggil objek layanan Amazon SES, meneruskan parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan `send` metode dalam pola `async/await`. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti **RECEIVER_ADDRESS** dengan alamat untuk mengirim email ke, dan **SENDER_ADDRESS** dengan *alamat* email untuk mengirim email dari.

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
```

```
import { sesClient } from "./libs/sesClient.js";

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

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

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. 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 `fileses_sendtemplatedemail.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

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 `SendTemplatedEmailCommand` metode kelas SES klien. Untuk memanggil `SendTemplatedEmailCommand` metode, panggil objek layanan klien Amazon SES, meneruskan parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan `send` metode dalam pola `async/await`. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti **REGION** dengan AWS Region Anda, **RECEIVER_ADDRESS** dengan alamat untuk mengirim email ke, **SENDER_ADDRESS** dengan alamat email untuk mengirim email dari, dan **TEMPLATE_NAME** dengan nama template.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the party
     gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
  });
};
```

```
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (err) {
    console.log("Failed to send template email", err);
    return err;
  }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. 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 `libs` direktori, dan buat modul Node.js dengan nama `filesesClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SES. Ganti **REGION** dengan AWS Region Anda.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileses_sendbulktemplatedemail.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk meneruskan nilai parameter yang menentukan email yang akan dikirim, termasuk alamat pengirim dan penerima, subjek, dan badan email dalam format teks biasa dan HTML, ke `SendBulkTemplatedEmailCommand` metode kelas SES klien. Untuk memanggil `SendBulkTemplatedEmailCommand` metode, panggil objek layanan Amazon SES, meneruskan parameter.

Note

Contoh ini mengimpor dan menggunakan klien paket AWS Service V3 yang diperlukan, perintah V3, dan menggunakan `send` metode dalam pola `async/await`. Anda dapat membuat contoh ini menggunakan perintah V2 sebagai gantinya dengan membuat beberapa perubahan kecil. Untuk detailnya, lihat [Menggunakan perintah V3](#).

Note

Ganti `RECEIVER_ADDRESSES` dengan alamat untuk mengirim email ke, dan `SENDER_ADDRESS` dengan *alamat* email untuk mengirim email dari.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");
```



```
const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map each
     user
     * to a 'Destination' and provide user specific replacement data to create
     personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (err) {
    console.log("Failed to send bulk template email", err);
  }
};
```

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt. Email diantrian untuk dikirim oleh Amazon SES.

```
node ses_sendbulktemplatedemail.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, AWS Lambda fungsi) mengkonsumsi atau menerima pesan atau pemberitahuan melalui salah satu protokol yang didukung (Amazon SQS, HTTP/S, email, AWS Lambda SMS,) ketika mereka berlangganan topik.

JavaScript API untuk Amazon SNS diekspos melalui [Kelas: 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 SNS klien berikut:

- [CreateTopicCommand](#)
- [ListTopicsCommand](#)
- [DeleteTopicCommand](#)
- [GetTopicAttributesCommand](#)
- [SetTopicAttributesCommand](#)

Tugas Prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#)

Membuat Topik

Dalam contoh ini, gunakan modul Node.js untuk membuat topik Amazon SNS.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filecreate-topic.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek untuk meneruskan topik baru ke `CreateTopicCommand` metode kelas SNS klien. Name Untuk memanggil `CreateTopicCommand` metode, buat fungsi asinkron yang menjalankan objek layanan Amazon SNS, melewati objek parameter. Yang data dikembalikan berisi ARN topik.

Note

Ganti *TOPIC_NAME* dengan nama topik.

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

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node create-topic.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filelist-topics.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek kosong untuk diteruskan ke `ListTopicsCommand` metode kelas SNS klien. Untuk memanggil `ListTopicsCommand` metode, buat fungsi asinkron yang menjalankan objek layanan Amazon SNS, melewati objek parameter. Yang data dikembalikan berisi larik topik Anda Amazon Resource Names (ARN).

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

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node list-topics.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus Topik

Dalam contoh ini, gunakan modul Node.js untuk menghapus topik Amazon SNS.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filedelete-topic.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek yang `TopicArn` berisi topik yang akan dihapus untuk diteruskan ke `DeleteTopicCommand` metode kelas SNS klien. Untuk memanggil `DeleteTopicCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

Note

Ganti **TOPIC_ARN** dengan Amazon Resource Name (ARN) dari topik yang Anda hapus.

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

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
```

```
const response = await snsClient.send(
  new DeleteTopicCommand({ TopicArn: topicArn }),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   }
// }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node delete-topic.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileget-topic-attributes.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang `TopicArn` berisi topik yang akan dihapus untuk diteruskan ke `GetTopicAttributesCommand` metode kelas SNS klien. Untuk memanggil `GetTopicAttributesCommand` metode, memanggil objek layanan klien Amazon SNS, meneruskan objek parameter.

Note

Ganti *TOPIC_ARN* dengan ARN topik.

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

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
```

```
//     DisplayName: '',
//     SubscriptionsDeleted: '1'
//   }
// }
return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node get-topic-attributes.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Mengatur Atribut Topik

Dalam contoh ini, gunakan modul Node.js untuk menyetel atribut yang dapat berubah dari topik Amazon SNS.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `set-topic-attributes.js`. Konfigurasi 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 `SetTopicAttributesCommand` metode kelas SNS klien. Untuk memanggil `SetTopicAttributesCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

Note

Ganti `ATTRIBUTE_NAME` dengan nama atribut yang Anda setel, `TOPIC_ARN` dengan Amazon Resource Name (ARN) dari topik yang atributnya ingin Anda tetapkan, dan `NEW_ATTRIBUTE_VALUE` dengan nilai baru untuk atribut tersebut.

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

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node set-topic-attributes.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 SNS klien ini:

- [PublishCommand](#)

Tugas Prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).

- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat. [JavaScript ES6/CommonJS sintaks](#)

Menerbitkan Pesan ke Topik SNS

Dalam contoh ini, gunakan modul Node.js untuk mempublikasikan pesan ke topik Amazon SNS.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filepublish-topic.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi parameter untuk menerbitkan pesan, termasuk teks pesan dan Amazon Resource Name (ARN) dari Amazon SnStoPic. Untuk detail tentang atribut SMS yang tersedia, lihat [SetSmSattributes](#).

Teruskan parameter ke `PublishCommand` metode kelas SNS klien. buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

Note

Ganti **MESSAGE_TEXT** dengan *teks* pesan, dan **TOPIC_ARN** dengan *ARN* dari topik SNS.

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

/**
```

```
* @param {string | Record<string, any>} message - The message to send. Can be a plain
string or an object
*
* if you are using the `json`
`MessageStructure`.
* @param {string} topicArn - The ARN of the topic to which you would like to publish.
*/
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node publish-topic.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 SNS klien berikut:

- [ListSubscriptionsByTopicCommand](#)
- [SubscribeCommand](#)
- [ConfirmSubscriptionCommand](#)
- [UnsubscribeCommand](#)

Tugas Prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).

- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat. [JavaScript ES6/CommonJS sintaks](#)

Daftar Langganan ke Topik

Dalam contoh ini, gunakan modul Node.js untuk mencantumkan semua langganan ke topik Amazon SNS.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filelist-subscriptions-by-topic.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi `TopicArn` parameter untuk topik yang langganannya ingin Anda daftarkan. Lewati parameter ke `ListSubscriptionsByTopicCommand` metode kelas SNS klien. Untuk memanggil `ListSubscriptionsByTopicCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, dan meneruskan objek parameter.

Note

Ganti **TOPIC_ARN** dengan Amazon Resource Name (ARN) untuk topik yang langganannya ingin Anda daftarkan.

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



```
/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node list-subscriptions-by-topic.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filesubscribe-email.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi `Protocol` parameter untuk menentukan email protokol, topik `TopicArn` untuk berlangganan, dan alamat email sebagai `messageEndpoint`. Lewati parameter ke `SubscribeCommand` metode kelas 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 `SubscribeCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, dan meneruskan objek parameter.

Note

Ganti **TOPIC_ARN** dengan Amazon Resource Name (ARN) untuk topik tersebut, dan **EMAIL_ADDRESS** dengan alamat email untuk berlangganan.

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

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 * subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
```

```
*/
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "user@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node subscribe-email.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Mengonfirmasi Langganan

Dalam contoh ini, gunakan modul Node.js untuk memverifikasi maksud pemilik endpoint untuk menerima email dengan memvalidasi token yang dikirim ke titik akhir dengan tindakan berlangganan sebelumnya.

Buat `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

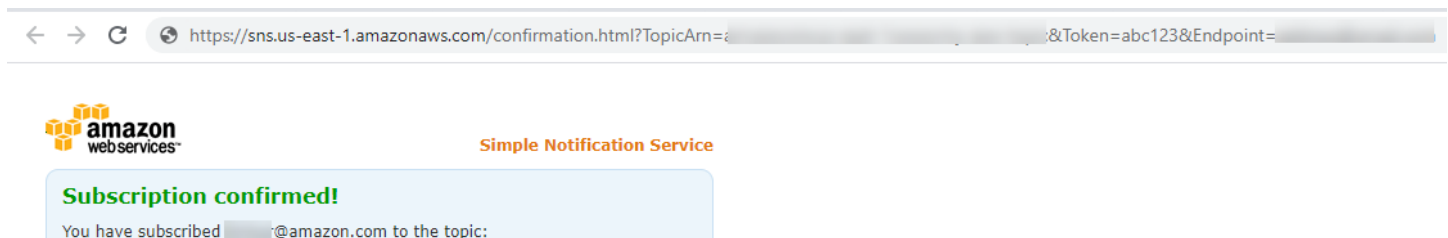
```
// The AWS Region can be provided here using the `region` property. If you leave it
  blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileconfirm-subscription.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Tentukan parameter, termasuk `TOPIC_ARN` dan `TOKEN`, dan tentukan nilai `TRUE` atau `FALSE` untuk `AuthenticateOnUnsubscribe`.

Token adalah token berumur pendek yang dikirim ke pemilik titik akhir selama tindakan sebelumnya `SUBSCRIBE`. Misalnya, untuk titik akhir email `TOKEN` ada di URL email Konfirmasi Langganan yang dikirim ke pemilik email. Misalnya, `abc123` adalah token di URL berikut.



Untuk memanggil `ConfirmSubscriptionCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

Note

Ganti `TOPIC_ARN` dengan Amazon Resource Name (ARN) untuk topik, `TOKEN` dengan nilai token dari URL yang dikirim ke pemilik titik akhir dalam `Subscribe` tindakan sebelumnya, dan tentukan `AuthenticateOnUnsubscribe` dengan nilai `TRUE` atau `FALSE`.

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

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 * that are not AWS services (HTTP/S, email) need to be
 * confirmed.
```

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

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node confirm-subscription.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti *REGION* dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filesubscribe-app.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal modul dan paket yang diperlukan.

Buat objek yang berisi `Protocol` parameter untuk menentukan `application` protokol, topik `TopicArn` untuk berlangganan, dan Nama Sumber Daya Amazon (ARN) dari titik akhir aplikasi seluler untuk parameter tersebut. Endpoint Lewati parameter ke `SubscribeCommand` metode kelas SNS klien.

Untuk memanggil `SubscribeCommand` metode, buat fungsi asinkron yang menjalankan objek layanan Amazon SNS, melewati objek parameter.

Note

Ganti *TOPIC_ARN* dengan Amazon Resource Name (ARN) untuk topik tersebut, dan *MOBILE_ENDPOINT_ARN* dengan titik akhir yang Anda berlangganan topik.

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

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 * created
 *
 * when an application registers for notifications.
 */
export const subscribeApp = async (
```

```
topicArn = "TOPIC_ARN",
endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node subscribe-app.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

```
// The AWS Region can be provided here using the `region` property. If you leave it
  blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filesubscribe-lambda.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya.

Buat objek yang berisi `Protocol` parameter, tentukan `lambda` protokol, topik `TopicArn` untuk berlangganan, dan `Amazon Resource Name (ARN)` dari fungsi AWS Lambda sebagai `Endpoint` parameter. Lewati parameter ke `SubscribeCommand` metode kelas SNS klien.

Untuk memanggil `SubscribeCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

Note

Ganti *TOPIC_ARN* dengan Amazon Resource Name (ARN) untuk topik tersebut, dan *LAMBDA_FUNCTION_ARN* dengan Amazon Resource Name (ARN) dari fungsi *Lambda*.

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

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  ),
```



```
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'pending confirmation'
// }
return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node subscribe-lambda.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileunsubscribe.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan.

Buat objek yang berisi `SubscriptionArn` parameter, tentukan Amazon Resource Name (ARN) dari langganan untuk berhenti berlangganan. Lewati parameter ke `UnsubscribeCommand` metode kelas SNS klien.

Untuk memanggil `UnsubscribeCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

Note

Ganti *TOPIC_SUBSCRIPTION_ARN* dengan Amazon Resource Name (ARN) dari langganan untuk berhenti berlangganan.

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

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
  xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node 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 SNS klien berikut:

- [GetSMSAttributesCommand](#)
- [SetSMSAttributesCommand](#)
- [CheckIfPhoneNumberIsOptedOutCommand](#)
- [ListPhoneNumbersOptedOutCommand](#)
- [PublishCommand](#)

Tugas Prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#)

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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
blank
```

```
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileget-sms-attributes.js`.

Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk mengunduh klien dan paket yang diperlukan. 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 [SetSmSattributes](#) di Referensi API Amazon Simple Notification Service.

Contoh ini mendapatkan `DefaultSMSType` atribut, yang mengontrol apakah pesan SMS dikirim sebagai `Promotional`, yang mengoptimalkan pengiriman pesan untuk menimbulkan biaya terendah, atau `Transactional`, yang mengoptimalkan pengiriman pesan untuk mencapai keandalan tertinggi. Lewati parameter ke `SetTopicAttributesCommand` metode kelas SNS klien. Untuk memanggil `SetSMSAttributesCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

Note

Ganti `ATTRIBUTE_NAME` dengan nama atribut.

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

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
```

```
//    cfId: undefined,  
//    attempts: 1,  
//    totalRetryDelay: 0  
//  },  
//  attributes: { DefaultSMSType: 'Transactional' }  
// }  
return response;  
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node get-sms-attributes.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `fileset-sms-attribute-type.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. 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 [SetSmSattributes](#) di Referensi API Amazon Simple Notification Service.

Contoh ini menetapkan `DefaultSMSType` atribut ke `Transactional`, yang mengoptimalkan pengiriman pesan untuk mencapai keandalan tertinggi. Lewati parameter

ke `SetTopicAttributesCommand` metode kelas SNS klien. Untuk memanggil `SetSMSAttributesCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

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

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional - (Default) Noncritical messages, such as marketing messages.
        // Transactional - Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node set-sms-attribute-type.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filecheck-if-phone-number-is-opted-out.js`. Konfigurasi 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 `CheckIfPhoneNumberIsOptedOutCommand` metode kelas SNS klien. Untuk memanggil `CheckIfPhoneNumberIsOptedOutCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

Note

1.

Ganti **PHONE_NUMBER** dengan nomor telepon.

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

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

export const checkIfPhoneNumberIsOptedOut = async (
```



```
phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node check-if-phone-number-is-opted-out.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
```

```
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filelist-phone-numbers-opted-out.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya. Buat objek kosong sebagai parameter.

Lewati objek ke `ListPhoneNumbersOptedOutCommand` metode kelas SNS klien. Untuk memanggil `ListPhoneNumbersOptedOutCommand` metode, buat fungsi asinkron yang menjalankan objek layanan klien Amazon SNS, melewati objek parameter.

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

export const listPhoneNumbersOptedOut = async () => {
  const response = await snsClient.send(
    new ListPhoneNumbersOptedOutCommand({}),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '44ff72fd-1037-5042-ad96-2fc16601df42',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   phoneNumbers: ['+15555550100']
  // }
  return response;
};
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node list-phone-numbers-opted-out.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 `libs` direktori, dan buat modul Node.js dengan nama `filesnsClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon SNS. Ganti **REGION** dengan AWS Region Anda.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filepublish-sms.js`. Konfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. 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 `+1001XXX5550100`.

Contoh ini menetapkan `PhoneNumber` parameter untuk menentukan nomor telepon untuk mengirim pesan. Lewati objek ke `PublishCommand` metode kelas SNS klien. Untuk memanggil `PublishCommand` metode, buat fungsi asinkron yang menjalankan objek layanan Amazon SNS, melewati objek parameter.

Note

Ganti **TEXT_MESSAGE** dengan pesan teks, dan **PHONE_NUMBER** dengan nomor telepon.

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

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 * string or an object
```

```

*                                     if you are using the `json`
`MessageStructure`.
* @param {*} phoneNumber - The phone number to send the message to.
*/
export const publish = async (
  message = "Hello from SNS!",
  phoneNumber = "+15555555555",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      // One of PhoneNumber, TopicArn, or TargetArn must be specified.
      PhoneNumber: phoneNumber,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '7410094f-efc7-5f52-af03-54737569ab77',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
  // }
  return response;
};
```

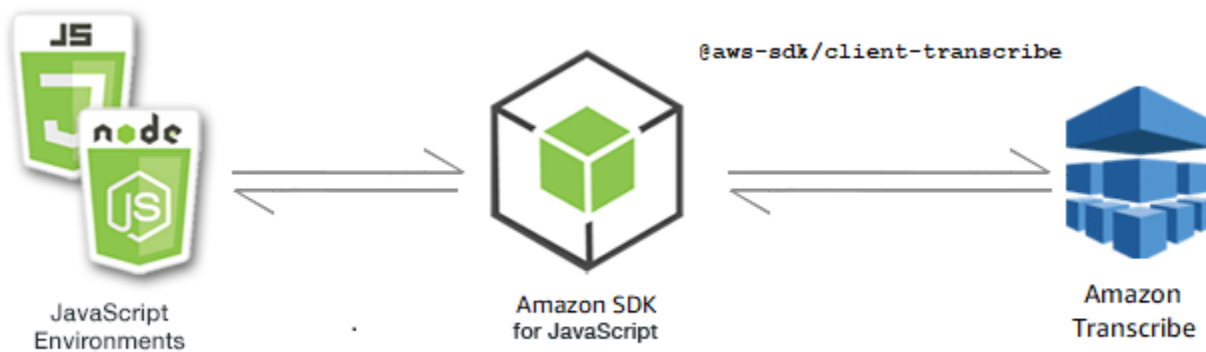
Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node publish-sms.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Contoh Amazon Transcribe

Amazon Transcribe memudahkan pengembang untuk menambahkan kemampuan ucapan ke teks ke aplikasi mereka.



JavaScript API untuk Amazon Transcribe diekspos melalui kelas [TranscribeService](#) klien.

Topik

- [Contoh Amazon Transcribe](#)
- [Contoh medis Amazon Transcribe](#)

Contoh Amazon Transcribe

Dalam contoh ini, serangkaian modul Node.js digunakan untuk membuat, membuat daftar, dan menghapus pekerjaan transkripsi menggunakan metode berikut dari kelas `TranscribeService` klien:

- [StartTranscriptionJobCommand](#)
- [ListTranscriptionJobsCommand](#)
- [DeleteTranscriptionJobCommand](#)

Untuk informasi selengkapnya tentang pengguna Amazon Transcribe, lihat panduan developer [Amazon Transcribe](#).

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#)

Memulai pekerjaan Amazon Transcribe

Contoh ini menunjukkan cara memulai pekerjaan transkripsi Amazon Transcribe menggunakan AWS SDK for JavaScript Untuk informasi lebih lanjut, lihat [StartTranscriptionJobCommand](#).

Buat `libs` direktori, dan buat modul Node.js dengan nama `filetranscribeClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Transcribe. Ganti **REGION** dengan AWS Region Anda.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filetranscribe-create-job.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Buat objek parameter, tentukan parameter yang diperlukan. Mulai pekerjaan menggunakan `StartMedicalTranscriptionJobCommand` perintah.

Note

Ganti *MEDICAL_JOB_NAME* dengan *nama* untuk pekerjaan transkripsi. Untuk *OUTPUT_BUCKET_NAME* tentukan bucket Amazon S3 tempat output disimpan. Untuk *JOB_TYPE* tentukan jenis pekerjaan. Untuk *SOURCE_LOCATION* tentukan lokasi file sumber. Untuk *SOURCE_FILE_LOCATION* tentukan lokasi file media input.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node transcribe-create-job.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Daftar pekerjaan Amazon Transcribe

Contoh ini menunjukkan caranya daftar pekerjaan transkripsi Amazon Transcribe menggunakan file. AWS SDK for JavaScript Untuk informasi selengkapnya tentang pengaturan lain yang dapat Anda ubah, lihat [ListTranscriptionJobCommand](#).

Buat `libs` direktori, dan buat modul Node.js dengan nama `filetranscribeClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Transcribe. Ganti **REGION** dengan AWS Region Anda.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filetranscribe-list-jobs.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Buat objek parameter dengan parameter yang diperlukan.

Note

Ganti **KEY_WORD** dengan kata kunci yang harus berisi nama pekerjaan yang dikembalikan.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};
```



```
export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node transcribe-list-jobs.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus pekerjaan Amazon Transcribe

Contoh ini menunjukkan cara menghapus pekerjaan transkripsi Amazon Transcribe menggunakan AWS SDK for JavaScript Untuk informasi lebih lanjut tentang opsional, lihat [DeleteTranscriptionJobCommand](#).

Buat `libs` direktori, dan buat modul Node.js dengan nama `filetranscribeClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Transcribe. Ganti **REGION** dengan AWS Region Anda.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filetranscribe-delete-job.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Tentukan AWS Wilayah, dan nama pekerjaan yang ingin Anda hapus.

Note

Ganti **JOB_NAME** dengan nama pekerjaan yang akan dihapus.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node transcribe-delete-job.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Contoh medis Amazon Transcribe

Dalam contoh ini, serangkaian modul Node.js digunakan untuk membuat, membuat daftar, dan menghapus pekerjaan transkripsi medis menggunakan metode berikut dari kelas `TranscribeService` klien:

- [StartMedicalTranscriptionJobCommand](#)

- [ListMedicalTranscriptionJobsCommand](#)
- [DeleteMedicalTranscriptionJobCommand](#)

Untuk informasi selengkapnya tentang pengguna Amazon Transcribe, lihat panduan developer [Amazon Transcribe](#).

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh-contoh ini menunjukkan cara mengimpor/mengekspor objek dan perintah layanan klien menggunakan ECMAScript6 (ES6).

- Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#).
- Jika Anda lebih suka menggunakan sintaks CommonJS, lihat [JavaScript ES6/CommonJS sintaks](#)

Memulai pekerjaan transkripsi medis Amazon Transcribe

Contoh ini menunjukkan cara memulai pekerjaan transkripsi medis Amazon Transcribe menggunakan AWS SDK for JavaScript Untuk informasi lebih lanjut, lihat [startMedicalTranscriptionJob](#).

Buat `libs` direktori, dan buat modul Node.js dengan nama `filetranscribeClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Transcribe. Ganti **REGION** dengan AWS Region Anda.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filetranscribe-create-medical-job.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Buat objek parameter, tentukan parameter yang diperlukan. Mulai pekerjaan medis menggunakan `StartMedicalTranscriptionJobCommand` perintah.

Note

Ganti *MEDICAL_JOB_NAME* dengan *nama* untuk pekerjaan transkripsi medis. Untuk *OUTPUT_BUCKET_NAME* tentukan bucket Amazon S3 tempat output disimpan. Untuk *JOB_TYPE* tentukan jenis pekerjaan. Untuk *SOURCE_LOCATION* tentukan lokasi file sumber. Untuk *SOURCE_FILE_LOCATION* tentukan lokasi file media input.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
    region
    // as the API endpoint that you are calling. For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};
```

```
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node transcribe-create-medical-job.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Daftar pekerjaan medis Amazon Transcribe

Contoh ini menunjukkan cara membuat daftar pekerjaan transkripsi Amazon Transcribe menggunakan AWS SDK for JavaScript Untuk informasi lebih lanjut, lihat [ListTranscriptionMedicalJobsCommand](#).


Buat `libs` direktori, dan buat modul Node.js dengan nama `filetranscribeClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Transcribe. Ganti **REGION** dengan AWS Region Anda.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filetranscribe-list-medical-jobs.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket

yang diperlukan. Buat objek parameter dengan parameter yang diperlukan, dan daftar pekerjaan medis menggunakan `ListMedicalTranscriptionJobsCommand` perintah.

 Note

Ganti **KEYWORD** dengan kata kunci yang harus berisi nama pekerjaan yang dikembalikan.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListMedicalTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Returns only transcription job names containing this
  string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListMedicalTranscriptionJobsCommand(params)
    );
    console.log("Success", data.MedicalTranscriptionJobName);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node transcribe-list-medical-jobs.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menghapus pekerjaan medis Amazon Transcribe

Contoh ini menunjukkan cara menghapus pekerjaan transkripsi Amazon Transcribe menggunakan AWS SDK for JavaScript Untuk informasi lebih lanjut tentang opsional, lihat [DeleteTranscriptionMedicalJobCommand](#).

Buat `libs` direktori, dan buat modul Node.js dengan nama `filetranscribeClient.js`. Salin dan tempel kode di bawah ini ke dalamnya, yang membuat objek klien Amazon Transcribe. Ganti **REGION** dengan AWS Region Anda.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat modul Node.js dengan nama `filetranscribe-delete-job.js`. Pastikan untuk mengonfigurasi SDK seperti yang ditunjukkan sebelumnya, termasuk menginstal klien dan paket yang diperlukan. Buat objek parameter dengan parameter yang diperlukan, dan hapus pekerjaan medis menggunakan `DeleteMedicalJobCommand` perintah.

Note

Ganti **JOB_NAME** dengan nama pekerjaan yang akan dihapus.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
  'medical_transcription_demo'
};

export const run = async () => {
```

```
try {
  const data = await transcribeClient.send(
    new DeleteMedicalTranscriptionJobCommand(params)
  );
  console.log("Success - deleted");
  return data; // For unit tests.
} catch (err) {
  console.log("Error", err);
}
};
run();
```

Untuk menjalankan contoh, masukkan yang berikut ini di command prompt.

```
node transcribe-delete-medical-job.js
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Menyiapkan Node.js pada instans Amazon EC2

Skenario umum untuk menggunakan Node.js dengan SDK for JavaScript adalah menyiapkan dan menjalankan aplikasi web Node.js pada instance Amazon Elastic Compute Cloud (Amazon EC2). 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 instans](#) di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Important

Gunakan Amazon Linux 2023 Amazon Machine Image (AMI) saat meluncurkan instans Amazon EC2 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 EC2 di Panduan Pengguna Amazon EC2](#) untuk Instans Linux.

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 sesi Amazon EC2 saat ini. Jika Anda memulai ulang sesi CLI Anda, Anda perlu menggunakan nvm lagi untuk mengaktifkan versi node yang diinstal. Jika instance dihentikan, Anda perlu menginstal node lagi. Alternatifnya adalah membuat Amazon Machine Image (AMI) dari instans Amazon EC2 setelah Anda memiliki konfigurasi yang ingin Anda simpan, seperti yang dijelaskan dalam topik berikut.

Membuat Gambar Mesin Amazon (AMI)

Setelah menginstal Node.js pada instans Amazon EC2, Anda dapat membuat Amazon Machine Image (AMI) dari instance tersebut. Membuat AMI memudahkan penyediaan beberapa instans Amazon EC2 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 Panduan Pengguna Amazon EC2 untuk Instans Linux.

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
- Node Package Manager (npm) —Lihat situs web [npm](#).

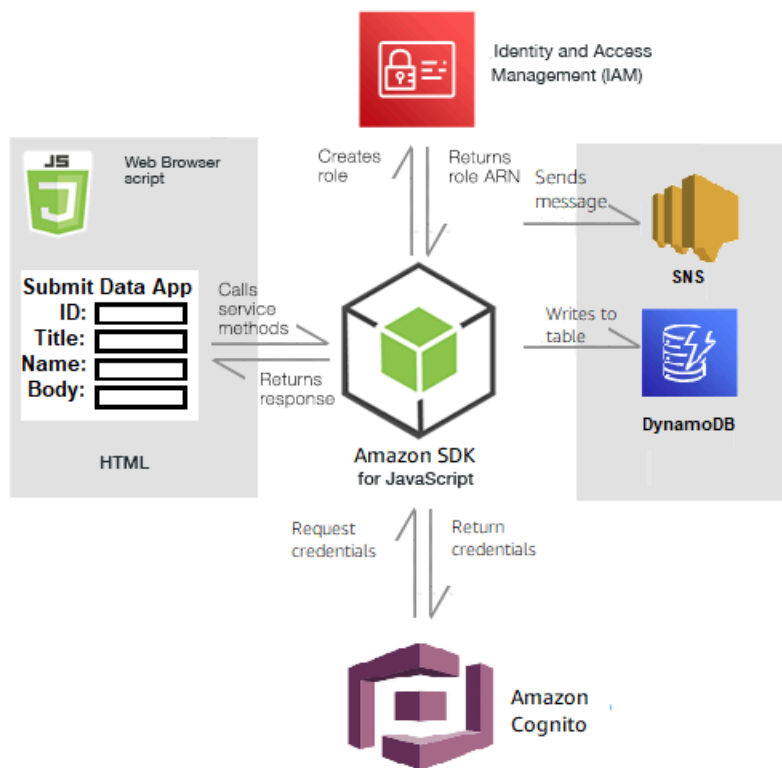
Membangun aplikasi untuk mengirimkan data ke DynamoDB

Tutorial Node.js lintas layanan ini menunjukkan cara membuat aplikasi yang memungkinkan pengguna mengirimkan data ke tabel Amazon DynamoDB. Aplikasi ini menggunakan layanan berikut:

- AWS Identity and Access Management(IAM) dan Amazon Cognito untuk otorisasi dan izin.
- Amazon DynamoDB (DynamoDB) untuk membuat dan memperbarui tabel.
- Amazon Simple Notification Service (Amazon SNS) untuk memberi tahu administrator aplikasi saat pengguna memperbarui tabel.

Skenario

Dalam tutorial ini, halaman HTML menyediakan aplikasi berbasis browser untuk mengirimkan data ke tabel Amazon DynamoDB. Aplikasi ini menggunakan Amazon SNS untuk memberi tahu administrator aplikasi saat pengguna memperbarui tabel.



Untuk membangun aplikasi:

1. [Prasyarat](#)
2. [Sumber daya penyediaan](#)
3. [Buat HTML](#)
4. [Buat skrip browser](#)
5. [Langkah selanjutnya](#)

Prasyarat

Selesaikan tugas prasyarat berikut:

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Buat sumber AWS daya

Aplikasi ini membutuhkan sumber daya berikut:

- AWS Identity and Access Management(IAM) Peran pengguna Amazon Cognito yang tidak diautentikasi dengan izin berikut:
 - SNS:Publish
 - dynamodb: PutItem
- Sebuah tabel DynamoDB.

Anda dapat membuat sumber daya ini secara manual di AWS konsol, tetapi kami sarankan untuk menyediakan sumber daya ini menggunakan AWS CloudFormation seperti yang dijelaskan dalam tutorial ini.

Buat sumber AWS daya menggunakan AWS CloudFormation

AWS CloudFormation memungkinkan Anda untuk membuat dan menyediakan penyebaran AWS infrastruktur yang dapat diprediksi dan berulang kali. Untuk informasi selengkapnya tentang AWS CloudFormation, lihat [AWS CloudFormation Panduan Pengguna](#).

Untuk membuat AWS CloudFormation tumpukan menggunakan AWS CLI:

1. Instal dan konfigurasi petunjuk AWS CLI berikut di [Panduan AWS CLI Pengguna](#).
2. Buat file bernama `setup.yaml` di direktori root folder proyek Anda, dan salin konten [di sini](#) [GitHub](#) ke dalamnya.

Note

AWS CloudFormationTemplate dibuat menggunakan yang AWS CDK tersedia [di sini GitHub](#). Untuk informasi selengkapnya tentang AWS CDK, lihat [Panduan Developer AWS Cloud Development Kit \(AWS CDK\)](#).

3. Jalankan perintah berikut dari baris perintah, ganti *STACK_NAME* dengan nama unik untuk tumpukan, dan *REGION di wilayah* Anda. AWS

Important

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM --region REGION
```

Untuk informasi selengkapnya tentang parameter create-stack perintah, lihat [panduan Referensi AWS CLI Perintah](#), dan [Panduan AWS CloudFormation Pengguna](#).

Untuk melihat sumber daya yang dibuat, buka AWS CloudFormation di konsol AWS manajemen, pilih tumpukan, dan pilih tab Sumber Daya.

4. Saat tumpukan dibuat, gunakan AWS SDK for JavaScript untuk mengisi tabel DynamoDB, seperti yang dijelaskan dalam. [Mengisi tabel](#)

Mengisi tabel

Untuk mengisi tabel, pertama membuat direktori bernama `libs`, dan di dalamnya membuat file bernama `dynamoClient.js`, dan paste konten di bawah ini ke dalamnya. Ganti *REGION* dengan AWS Region Anda, dan ganti *IDENTITY_POOL_ID* dengan ID Kumpulan Identitas Amazon Cognito. Ini menciptakan objek klien DynamoDB.

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

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

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

export { dynamoClient };
```

Kode ini tersedia [di sini GitHub](#).

Selanjutnya, buat `dynamoAppHelperFiles` folder di folder proyek Anda, buat file `update-table.js` di dalamnya, dan salin konten [di sini GitHub](#) ke dalamnya.

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

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

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

```
};  
run();
```

Jalankan perintah berikut dari baris perintah.

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

Kode ini tersedia [di sini GitHub](#).

Buat halaman front-end untuk aplikasi

Di sini Anda membuat halaman browser HTML front-end untuk aplikasi.

Buat DynamoDBApp direktori, buat file bernama `index.html`, dan salin kode dari [sini GitHub](#). `scriptElemen` menambahkan `main.js` file, yang berisi semua yang diperlukan JavaScript untuk contoh. Anda akan membuat `main.js` file nanti dalam tutorial ini. Kode yang tersisa di `index.html` membuat halaman browser yang menangkap data yang dimasukkan pengguna.

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Buat skrip browser

Pertama, buat objek klien layanan yang diperlukan untuk contoh. Buat `libs` direktori, buat `snsClient.js`, dan tempel kode di bawah ini ke dalamnya. Ganti **REGION** dan **IDENTITY_POOL_ID** di masing-masing.

Note

Gunakan ID kumpulan identitas Amazon Cognito yang Anda buat. [Buat sumber AWS daya](#)

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";  
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";  
import { SNSClient } from "@aws-sdk/client-sns";  
  
const REGION = "REGION";  
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.  
  
// Create an Amazon Comprehend service client object.
```

```
const snsClient = new SNSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { snsClient };
```

Kode ini tersedia [di sini GitHub](#) .

Untuk membuat skrip browser untuk contoh ini, dalam folder bernama `DynamoDBApp`, buat modul Node.js dengan nama file `add_data.js` dan tempel kode di bawah ini ke dalamnya. `submitData` fungsi mengirimkan data ke tabel DynamoDB, dan mengirimkan teks SMS ke administrator aplikasi menggunakan Amazon SNS.

Dalam `submitData` fungsi tersebut, deklarasikan variabel untuk nomor telepon target, nilai yang dimasukkan pada antarmuka aplikasi, dan untuk nama bucket Amazon S3. Selanjutnya, buat objek parameter untuk menambahkan item ke tabel. Jika tidak ada nilai yang kosong, `submitData` tambahkan item ke tabel, dan kirim pesan. Ingatlah untuk membuat fungsi tersedia untuk browser, dengan `window.submitData = submitData`.

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

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

  //Set the parameters for the table
  const params = {
    TableName: tableName,
```



```
// Define the attributes and values of the item to be added. Adding ' + "' '
converts a value to
// a string.
Item: {
  id: { N: id + "' },
  title: { S: title + "' },
  name: { S: name + "' },
  body: { S: body + "' },
},
};
// Check that all the fields are completed.
if (id !== "" && title !== "" && name !== "" && body !== "") {
  try {
    //Upload the item to the table
    await dynamoClient.send(new PutItemCommand(params));
    alert("Data added to table.");
    try {
      // Create the message parameters object.
      const messageParams = {
        Message: "A new item with ID value was added to the DynamoDB",
        PhoneNumber: "PHONE_NUMBER", //PHONE_NUMBER, in the E.164 phone number
        structure.
        // For example, ak standard local formatted number, such as (415) 555-2671,
        is +14155552671 in E.164
        // format, where '1' in the country code.
      };
      // Send the SNS message
      const data = await snsClient.send(new PublishCommand(messageParams));
      console.log(
        "Success, message published. MessageID is " + data.MessageId,
      );
    } catch (err) {
      // Display error message if error is not sent
      console.error(err, err.stack);
    }
  } catch (err) {
    // Display error message if item is no added to table
    console.error(
      "An error occurred. Check the console for further information",
      err,
    );
  }
  // Display alert if all field are not completed.
} else {
```

```
    alert("Enter data in each field.");
  }
};
// Expose the function to the browser
window.submitData = submitData;
```

Kode contoh ini dapat ditemukan [di sini GitHub](#).

Akhirnya, jalankan yang berikut ini pada command prompt untuk bundel JavaScript untuk contoh ini dalam file bernama `main.js`:

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

Note

Untuk informasi tentang menginstal webpack, lihat [Bundel aplikasi dengan webpack](#).

Untuk menjalankan aplikasi, buka `index.html` di browser Anda.

Hapus sumber daya

Seperti yang dinyatakan di awal tutorial ini, pastikan untuk menghentikan semua sumber daya yang Anda buat saat melalui tutorial ini untuk memastikan bahwa Anda tidak dikenakan biaya. Anda dapat melakukan ini dengan menghapus AWS CloudFormation tumpukan yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini, sebagai berikut:

1. Buka [AWS CloudFormation di konsol AWS manajemen](#).
2. Buka halaman Stacks, dan pilih tumpukan.
3. Pilih Hapus.

Untuk contoh AWS lintas layanan lainnya, lihat contoh [AWS SDK for JavaScript lintas layanan](#).

Membangun aplikasi transkripsi dengan pengguna yang diautentikasi

Dalam tutorial ini, Anda belajar cara:

- Menerapkan otentikasi menggunakan kumpulan identitas Amazon Cognito untuk menerima pengguna yang digabungkan dengan kumpulan pengguna Amazon Cognito.

- Gunakan Amazon Transcribe untuk mentranskripsikan dan menampilkan rekaman suara di browser.

Skenario

Aplikasi ini memungkinkan pengguna untuk mendaftar dengan email dan nama pengguna yang unik. Pada konfirmasi email mereka, mereka dapat merekam pesan suara yang secara otomatis ditranskripsikan dan ditampilkan di aplikasi.

Cara kerjanya

Aplikasi ini menggunakan dua bucket Amazon S3, satu untuk meng-host kode aplikasi, dan satu lagi untuk menyimpan transkripsi. Aplikasi ini menggunakan kumpulan pengguna Amazon Cognito untuk mengautentikasi pengguna Anda. Pengguna yang diautentikasi memiliki izin IAM untuk mengakses layanan yang diperlukan. AWS

Saat pertama kali pengguna merekam pesan suara, Amazon S3 membuat folder unik dengan nama pengguna di bucket Amazon S3 untuk menyimpan transkripsi. Amazon Transcribe mentranskripsikan pesan suara ke teks, dan menyimpannya di JSON di folder pengguna. Saat pengguna menyegarkan aplikasi, transkripsi mereka ditampilkan dan tersedia untuk diunduh atau dihapus.

Tutorial harus memerlukan waktu sekitar 30 menit untuk menyelesaikannya.

Langkah-langkah

Untuk membangun aplikasi:

1. [Prasyarat](#)
2. [Buat sumber AWS daya](#)
3. [Buat HTML](#)
4. [Siapkan skrip browser](#)
5. [Jalankan aplikasi](#)
6. [Hapus sumber daya](#)

Prasyarat

- Siapkan lingkungan proyek untuk menjalankan JavaScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#). Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Buat sumber AWS daya

Bagian ini menjelaskan cara menyediakan AWS sumber daya untuk aplikasi ini menggunakan AWS Cloud Development Kit (AWS CDK).

Note

AWS CDK ini adalah kerangka pengembangan perangkat lunak yang memungkinkan Anda untuk menentukan sumber daya aplikasi cloud. Untuk informasi selengkapnya, lihat [Panduan Developer AWS Cloud Development Kit \(AWS CDK\)](#).

Untuk membuat sumber daya untuk aplikasi, gunakan templat [di sini GitHub](#) untuk membuat AWS CDK tumpukan menggunakan [Konsol Manajemen Layanan AWS Web](#) atau [AWS CLI](#). Untuk petunjuk tentang cara memodifikasi tumpukan, atau untuk menghapus tumpukan dan sumber daya terkait ketika Anda telah menyelesaikan tutorial, lihat [di sini GitHub](#).

Note

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

Tumpukan yang dihasilkan secara otomatis menyediakan sumber daya berikut.

- Kumpulan identitas Amazon Cognito dengan peran pengguna yang diautentikasi.

- Kebijakan IAM dengan izin untuk Amazon S3 dan Amazon Transcribe dilampirkan ke peran pengguna yang diautentikasi.
- Kumpulan pengguna Amazon Cognito yang memungkinkan pengguna untuk mendaftar dan masuk ke aplikasi.
- Bucket Amazon S3 untuk meng-host file aplikasi.
- Ember Amazon S3 untuk menyimpan transkripsi.

Important

Bucket Amazon S3 ini memungkinkan akses publik BACA (DAFTAR), yang memungkinkan siapa saja untuk membuat daftar objek di dalam ember dan berpotensi menyalahgunakan informasi. Jika Anda tidak menghapus bucket Amazon S3 ini segera setelah menyelesaikan tutorial, kami sangat menyarankan Anda mematuhi [Praktik Terbaik Keamanan di Amazon S3 di](#) Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Buat HTML

Buat `index.html` file, dan salin dan tempel konten di bawah ini ke dalamnya. Halaman ini menampilkan panel tombol untuk merekam pesan suara, dan tabel yang menampilkan pesan yang sebelumnya ditranskripsikan pengguna saat ini. Tag skrip di akhir body elemen memanggil `main.js`, yang berisi semua skrip browser untuk aplikasi. Anda membuat `main.js` menggunakan Webpack, seperti yang dijelaskan di bagian berikut dari tutorial ini.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>title</title>
  <link rel="stylesheet" type="text/css" href="recorder.css">
  <style>
    table, td {
      border: 1px solid black;
    }
  </style>
</head>
<body>
<h2>Record</h2>
```

```
<p>
  <button id="record" onclick="startRecord()"></button>
  <button id="stopRecord" disabled onclick="stopRecord()">Stop</button>
<p id="demo" style="visibility: hidden;"></p>
</p>
<p>
  <audio id="recordedAudio"></audio>
</p>

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

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

</html>
```

Contoh kode ini tersedia [di sini GitHub](#).

Siapkan skrip browser

Ada tiga file, `index.html`, `recorder.js`, dan `helper.js`, yang harus Anda bundel menjadi satu `main.js` menggunakan Webpack. Bagian ini menjelaskan secara rinci hanya fungsi `index.js` yang menggunakan SDK for JavaScript, yang tersedia [di GitHub sini](#).

Note

`recorder.js` dan `helper.js` diperlukan tetapi, karena tidak mengandung kode Node.js, dijelaskan dalam komentar sebaris di [sini](#) dan [di sini](#) masing-masing. GitHub

Pertama, tentukan parameternya. `COGNITO_ID` adalah titik akhir untuk Kumpulan Pengguna Amazon Cognito yang Anda buat dalam topik [Buat sumber AWS daya](#) tutorial ini. Itu diformat. `cognito-`

`idp.AWS_REGION.amazonaws.com/USER_POOL_ID` Id kumpulan pengguna adalah `ID_TOKEN` dalam token AWS kredensial, yang dilucuti dari URL aplikasi oleh `getToken` fungsi dalam file `'helper.js'`. Token ini diteruskan ke `loginData` variabel, yang menyediakan objek klien Amazon Transcribe dan Amazon S3 dengan login. Ganti `"REGION"` dengan AWS Region, dan `"BUCKET"` dengan Ganti `"IDENTITY_POOL_ID"` dengan halaman `from Sample IdentityPoolId` dari kumpulan identitas Amazon Cognito yang Anda buat untuk contoh ini. Ini juga diteruskan ke setiap objek klien.

```
// Import the required AWS SDK clients and commands for Node.js
import "./helper.js";
import "./recorder.js";
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import {
  CognitoIdentityProviderClient,
  GetUserCommand,
} from "@aws-sdk/client-cognito-identity-provider";
import { S3RequestPresigner } from "@aws-sdk/s3-request-presigner";
import { createRequest } from "@aws-sdk/util-create-request";
import { formatUrl } from "@aws-sdk/util-format-url";
import {
  TranscribeClient,
  StartTranscriptionJobCommand,
} from "@aws-sdk/client-transcribe";
import {
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
  ListObjectsCommand,
  DeleteObjectCommand,
} from "@aws-sdk/client-s3";
import fetch from "node-fetch";

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

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

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

// Create an Amazon S3 client object.
const s3Client = new S3Client({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});
```

Saat halaman HTML dimuat, akan `updateUserInterface` membuat folder dengan nama pengguna di bucket Amazon S3 jika ini pertama kalinya mereka masuk ke aplikasi. Jika tidak, itu memperbarui antarmuka pengguna dengan transkrip apa pun dari sesi pengguna sebelumnya.

```
window.onload = async () => {
  // Set the parameters.
  const userParams = {
    // Get the access token. 'GetAccessToken()' is in 'helper.js'.
    AccessToken: getAccessToken(),
  };
  // Create a CognitoIdentityProviderClient client object.
  const client = new CognitoIdentityProviderClient({ region: params.Region });
  try {
    const data = await client.send(new GetUserCommand(userParams));
```



```
const username = data.Username;
// Export username for use in 'recorder.js'.
exports.username = username;
try {
  // If this is user's first sign-in, create a folder with user's name in Amazon S3
  bucket.
  // Otherwise, no effect.
  const Key = `${username}/`;
  try {
    const data = await s3Client.send(
      new PutObjectCommand({ Key: Key, Bucket: params.Bucket })
    );
    console.log("Folder created for user ", data.Username);
  } catch (err) {
    console.log("Error", err);
  }
  try {
    // Get a list of the objects in the Amazon S3 bucket.
    const data = await s3Client.send(
      new ListObjectsCommand({ Bucket: params.Bucket, Prefix: username })
    );
    // Create a variable for the list of objects in the Amazon S3 bucket.
    const output = data.Contents;
    // Loop through the objects, populating a row on the user interface for each
    object.
    for (var i = 0; i < output.length; i++) {
      var obj = output[i];
      const objectParams = {
        Bucket: params.Bucket,
        Key: obj.Key,
      };
      // Get the name of the object from the Amazon S3 bucket.
      const data = await s3Client.send(new GetObjectCommand(objectParams));
      // Extract the body contents, a readable stream, from the returned data.
      const result = data.Body;
      // Create a variable for the string version of the readable stream.
      let stringResult = "";
      // Use 'yieldUnit8Chunks' to convert the readable streams into JSON.
      for await (let chunk of yieldUnit8Chunks(result)) {
        stringResult += String.fromCharCode.apply(null, chunk);
      }
      // The setTimeout function waits while readable stream is converted into
      JSON.
      setTimeout(function () {
```

```
        // Parse JSON into human readable transcript, which will be displayed on
user interface (UI).
        const outputJSON =
            JSON.parse(stringResult).results.transcripts[0].transcript;
        // Create name for transcript, which will be displayed.
        const outputJSONTime = JSON.parse(stringResult)
            .jobName.split("/")[0]
            .replace("-job", "");
        i++;
        //
        // Display the details for the transcription on the UI.
        // 'displayTranscriptionDetails()' is in 'helper.js'.
        displayTranscriptionDetails(
            i,
            outputJSONTime,
            objectParams.Key,
            outputJSON
        );
    }, 1000);
}
} catch (err) {
    console.log("Error", err);
}
} catch (err) {
    console.log("Error creating presigned URL", err);
}
} catch (err) {
    console.log("Error", err);
}
};

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

Saat pengguna merekam pesan suara untuk transkripsi, akan upload mengunggah rekaman ke bucket Amazon S3. Fungsi ini dipanggil dari `recorder.js` file.

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

  // Create a presigned URL to upload the transcription to the Amazon S3 bucket when it
  // is ready.
  try {
    // Create an Amazon S3RequestPresigner object.
    const signer = new S3RequestPresigner({ ...s3Client.config });
    // Create the request.
    const request = await createRequest(
      s3Client,
      new PutObjectCommand({ Key, Bucket: params.Bucket })
    );
    // Define the duration until expiration of the presigned URL.
    const expiration = new Date(Date.now() + 60 * 60 * 1000);
    // Create and format the presigned URL.
    signedUrl = formatUrl(await signer.presign(request, expiration));
    console.log(`\nPutting "${Key}"`);
  } catch (err) {
    console.log("Error creating presigned URL", err);
  }
  try {
    // Upload the object to the Amazon S3 bucket using a presigned URL.
    response = await fetch(signedUrl, {
      method: "PUT",
      headers: {
        "content-type": "application/octet-stream",
      },
      body: blob,
    });
    // Create the transcription job name. In this case, it's the current date and time.
    const today = new Date();
    const date =
      today.getFullYear() +
```

```
    "-" +
    (today.getMonth() + 1) +
    "-" +
    today.getDate();
const time =
    today.getHours() + "-" + today.getMinutes() + "-" + today.getSeconds();
const jobName = date + "-time-" + time;

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

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

`deleteTranscription` menghapus transkripsi dari antarmuka pengguna, dan `deleteRow` menghapus transkripsi yang ada dari bucket Amazon S3. Keduanya dipicu oleh tombol Delete pada antarmuka pengguna.

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

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

Akhirnya, jalankan yang berikut ini pada command prompt untuk bundel JavaScript untuk contoh ini dalam file bernama `main.js`:

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

Note

Untuk informasi tentang menginstal webpack, lihat [Bundel aplikasi dengan webpack](#).

Jalankan aplikasi

Anda dapat melihat aplikasi di lokasi di bawah ini.

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

Amazon Cognito memudahkan menjalankan aplikasi dengan menyediakan tautan di Konsol Manajemen Layanan AWS Web. Cukup navigasikan ke pengaturan klien Aplikasi dari kumpulan pengguna Amazon Cognito Anda, dan pilih Launch Hosted UI. URL untuk aplikasi memiliki format berikut.

Important

UI yang Dihosting secara default adalah tipe respons 'kode'. Namun, tutorial ini dirancang untuk tipe respons 'token', jadi Anda harus mengubahnya.

Hapus AWS sumber daya

Ketika Anda menyelesaikan tutorial, Anda harus menghapus sumber daya sehingga Anda tidak dikenakan biaya yang tidak perlu. Karena Anda menambahkan konten ke kedua bucket Amazon S3, Anda harus menghapusnya secara manual. Kemudian Anda dapat menghapus sumber daya yang tersisa menggunakan [Konsol Manajemen Layanan AWS Web](#) atau [AWS CLI](#). Petunjuk tentang cara memodifikasi tumpukan, atau untuk menghapus tumpukan dan sumber daya terkait ketika Anda telah menyelesaikan tutorial, lihat [di sini GitHub](#).

Memanggil Lambda dengan API Gateway

Anda dapat menjalankan fungsi Lambda dengan menggunakan Amazon API Gateway, yang AWS merupakan layanan untuk membuat, menerbitkan, memelihara, memantau, dan mengamankan REST, HTTP, WebSocket dan API dalam skala besar. Pengembang API dapat membuat API yang mengakses AWS atau layanan web lainnya, serta data yang disimpan di dalam AWS Cloud. Sebagai pengembang API Gateway, Anda dapat membuat API untuk digunakan dalam aplikasi klien Anda sendiri. Untuk informasi selengkapnya, lihat [Apa itu Amazon API Gateway](#).

AWS Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Anda dapat membuat fungsi Lambda dalam berbagai bahasa pemrograman. Untuk informasi selengkapnya tentang AWS Lambda, lihat [Apa itu AWS Lambda](#).

Dalam contoh ini, Anda membuat fungsi Lambda menggunakan API runtime JavaScript Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu.

Misalnya, asumsikan bahwa organisasi mengirim pesan teks seluler kepada karyawannya yang memberi selamat kepada mereka pada tanggal ulang tahun satu tahun, seperti yang ditunjukkan dalam ilustrasi ini.



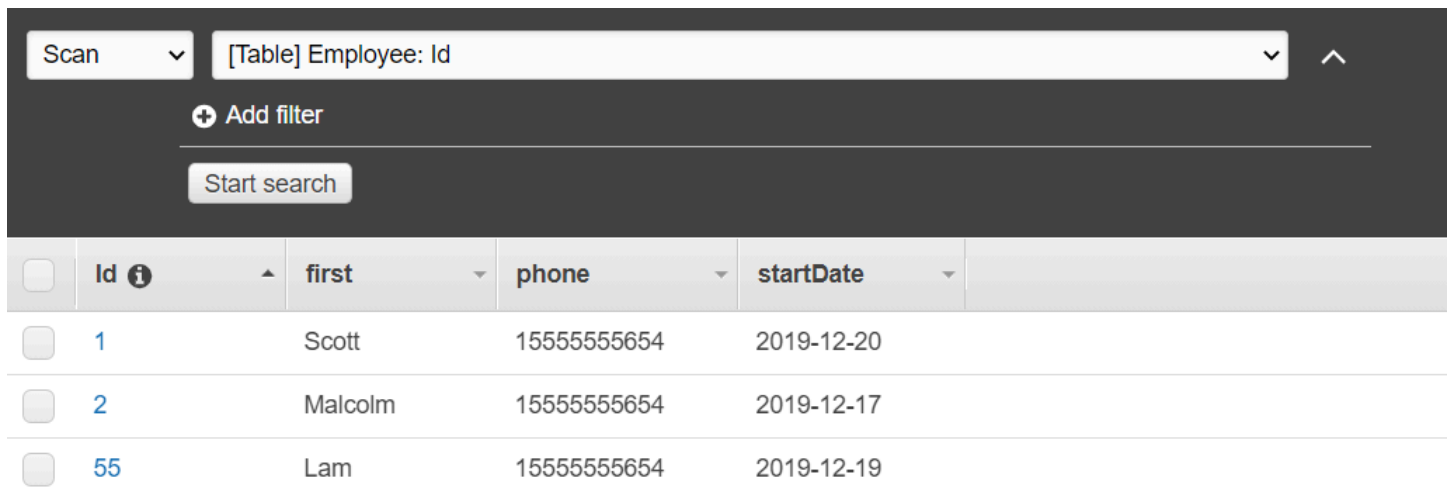
Contoh harus memakan waktu sekitar 20 menit untuk menyelesaikannya.

Contoh ini menunjukkan cara menggunakan JavaScript logika untuk membuat solusi yang melakukan kasus penggunaan ini. Misalnya, Anda akan mempelajari cara membaca database untuk menentukan karyawan mana yang telah mencapai tanggal ulang tahun satu tahun, cara memproses data, dan mengirim pesan teks semuanya dengan menggunakan fungsi Lambda. Kemudian Anda akan belajar cara menggunakan API Gateway untuk menjalankan AWS Lambda fungsi ini dengan menggunakan endpoint Istirahat. Misalnya, Anda dapat menjalankan fungsi Lambda dengan menggunakan perintah curl ini:

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/employee"
```

AWSTutorial ini menggunakan tabel Amazon DynamoDB bernama Employee yang berisi bidang-bidang ini.

- id - kunci utama untuk tabel.
- FirstName - nama depan karyawan.
- telepon - nomor telepon karyawan.
- StartDate - tanggal mulai karyawan.



	Id <i>i</i>	first	phone	startDate
<input type="checkbox"/>	1	Scott	15555555654	2019-12-20
<input type="checkbox"/>	2	Malcolm	15555555654	2019-12-17
<input type="checkbox"/>	55	Lam	15555555654	2019-12-19

Important

Biaya untuk menyelesaikan: AWS Layanan yang termasuk dalam dokumen ini termasuk dalam Tingkat AWS Gratis. Namun, pastikan untuk menghentikan semua sumber daya setelah Anda menyelesaikan contoh ini untuk memastikan bahwa Anda tidak dikenakan biaya.

Untuk membangun aplikasi:

1. [Prasyarat lengkap](#)
2. [Buat sumber AWS daya](#)
3. [Siapkan skrip browser](#)
4. [Buat dan unggah fungsi Lambda](#)
5. [Menyebarkan fungsi Lambda](#)
6. [Jalankan aplikasi](#)
7. [Hapus sumber daya](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).

- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Buat sumber AWS daya

Tutorial ini membutuhkan sumber daya berikut:

- Tabel Amazon DynamoDB Employee bernama dengan kunci Id bernama dan bidang yang ditunjukkan pada ilustrasi sebelumnya. Pastikan Anda memasukkan data yang benar, termasuk ponsel yang valid yang ingin Anda uji kasus penggunaan ini. Untuk informasi selengkapnya, lihat [Membuat Tabel](#).
- Peran IAM dengan izin terlampir untuk menjalankan fungsi Lambda.
- Bucket Amazon S3 untuk menampung fungsi Lambda.

Anda dapat membuat sumber daya ini secara manual, tetapi kami merekomendasikan penyediaan sumber daya ini menggunakan AWS CloudFormation seperti yang dijelaskan dalam tutorial ini.

Buat sumber AWS daya menggunakan AWS CloudFormation

AWS CloudFormation memungkinkan Anda untuk membuat dan menyediakan penyebaran AWS infrastruktur yang dapat diprediksi dan berulang kali. Untuk informasi selengkapnya tentang AWS CloudFormation, lihat [AWS CloudFormation Panduan Pengguna](#).

Untuk membuat AWS CloudFormation tumpukan menggunakan AWS CLI:

1. Instal dan konfigurasi petunjuk AWS CLI berikut di [Panduan AWS CLI Pengguna](#).
2. Buat file bernama `setup.yaml` di direktori root folder proyek Anda, dan salin konten [di sini](#) [GitHub](#) ke dalamnya.

Note

AWS CloudFormationTemplate dibuat menggunakan yang AWS CDK tersedia [di sini](#) [GitHub](#). Untuk informasi selengkapnya tentang AWS CDK, lihat [Panduan Developer AWS Cloud Development Kit \(AWS CDK\)](#).

3. Jalankan perintah berikut dari baris perintah, ganti `STACK_NAME` dengan *nama* unik untuk tumpukan.

⚠ Important

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

Untuk informasi selengkapnya tentang parameter `create-stack` perintah, lihat [panduan Referensi AWS CLI Perintah](#), dan [Panduan AWS CloudFormation Pengguna](#).

4. Selanjutnya, isi tabel dengan mengikuti prosedur [Mengisi tabel](#).

Mengisi tabel

Untuk mengisi tabel, pertama membuat direktori bernama `libs`, dan di dalamnya membuat file bernama `dynamoClient.js`, dan paste konten di bawah ini ke dalamnya.

```
const { DynamoDBClient } = require ( "@aws-sdk/client-dynamodb" );
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon Lambda service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

Kode ini tersedia [di sini GitHub](#).

Selanjutnya, buat file bernama `populate-table.js` di direktori root folder proyek Anda, dan salin konten [di sini GitHub](#) ke dalamnya. Untuk salah satu item, ganti nilai `phone` properti dengan nomor ponsel yang valid dalam format E.164, dan nilai untuk `startDate` dengan tanggal hari ini.

Jalankan perintah berikut dari baris perintah.

```
node populate-table.js
```

```
const { BatchWriteItemCommand } = require ( "aws-sdk/client-dynamodb" );
```

```
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Set the parameters.
export const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  }
};
```

```
    } catch (err) {  
      console.log("Error", err);  
    }  
  };  
  run();
```

Kode ini tersedia [di sini GitHub](#).

Membuat AWS Lambda fungsi

Mengkonfigurasi SDK

Di `libs` direktori, buat file bernama `snsClient.js` dan `lambdaClient.js`, dan tempelkan konten di bawah ini ke dalam file-file ini, masing-masing.

```
const { SNSClient } = require ( "@aws-sdk/client-sns" );  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon SNS service client object.  
const snsClient = new SNSClient({ region: REGION });  
module.exports = { snsClient };
```

Ganti **REGION** dengan AWS Region. Kode ini tersedia [di sini GitHub](#).

```
const { LambdaClient } = require ( "@aws-sdk/client-lambda" );  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon Lambda service client object.  
const lambdaClient = new LambdaClient({ region: REGION });  
module.exports = { lambdaClient };
```

Ganti **REGION** dengan AWS Region. Kode ini tersedia [di sini GitHub](#).

Pertama, impor modul dan perintah yang diperlukan AWS SDK for JavaScript (v3). Kemudian hitung tanggal hari ini dan tetapkan ke parameter. Ketiga, buat parameter untuk `ScanCommand`. Ganti **TABLE_NAME** dengan *nama* tabel yang Anda buat di [Buat sumber AWS daya](#) bagian contoh ini.

Cuplikan kode berikut menunjukkan langkah ini. (Lihat [Bundling fungsi Lambda](#) contoh lengkapnya.)

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

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

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
  want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "Employees",
};
```

Memindai tabel DynamoDB

Pertama, buat fungsi `async/await` yang dipanggil `sendText` untuk mempublikasikan pesan teks menggunakan Amazon SNS. `PublishCommand` Kemudian, tambahkan pola `try` blok yang memindai tabel DynamoDB untuk karyawan dengan ulang tahun kerja mereka hari ini, dan kemudian memanggil fungsi untuk mengirim pesan `sendText` teks kepada karyawan ini. Jika terjadi kesalahan, `catch` blok dipanggil.

Cuplikan kode berikut menunjukkan langkah ini. (Lihat [Bundling fungsi Lambda](#) contoh lengkapnya.)

```
// Helper function to send message using Amazon SNS.
exports.handler = async () => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
```

```
    await snsClient.send(new PublishCommand(textParams));
    console.log("Message sent");
  } catch (err) {
    console.log("Error, message not sent ", err);
  }
}
try {
  // Scan the table to identify employees with work anniversary today.
  const data = await dynamoClient.send(new ScanCommand(params));
  data.Items.forEach(function (element) {
    const textParams = {
      PhoneNumber: element.phone.N,
      Message:
        "Hi " +
        element.firstName.S +
        "; congratulations on your work anniversary!",
    };
    // Send message using Amazon SNS.
    sendText(textParams);
  });
} catch (err) {
  console.log("Error, could not scan table ", err);
}
};
```

Bundling fungsi Lambda

Topik ini menjelaskan cara menggabungkan `mylambdafunction.ts` dan AWS SDK for JavaScript modul yang diperlukan untuk contoh ini ke dalam file yang dibundel yang disebut `index.js`.

1. Jika Anda belum melakukannya, ikuti contoh ini [Tugas prasyarat](#) untuk menginstal webpack.

Note

Untuk informasi tentang webpack, lihat [Bundel aplikasi dengan webpack](#).

2. Jalankan yang berikut ini di baris perintah JavaScript untuk menggabungkan contoh ini ke dalam file bernama `<index.js>`:

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

⚠ Important

Perhatikan output diberi nama `index.js`. Ini karena fungsi Lambda harus memiliki `index.js` handler untuk bekerja.

3. Kompres file output yang dibundel, `index.js`, ke dalam file ZIP bernama `mylambdafunction.zip`.
4. Unggah `mylambdafunction.zip` ke bucket Amazon S3 yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini.

Deploy fungsi Lambda

Di root proyek Anda, buat `lambda-function-setup.ts` file, dan tempel konten di bawah ini ke dalamnya.

Ganti `BUCKET_NAME` dengan nama bucket Amazon S3 tempat Anda mengunggah versi ZIP fungsi Lambda Anda. Ganti `ZIP_FILE_NAME` dengan `nama nama` versi ZIP dari fungsi Lambda Anda. Ganti `ROLE` dengan Amazon Resource Number (ARN) dari peran IAM yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini. Ganti `LAMBDA_FUNCTION_NAME` dengan `nama` untuk fungsi Lambda.

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand
} = require ( "@aws-sdk/client-lambda" );
const { lambdaClient } = require ( "../libs/lambdaClient.js" );

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
```

```
"Scans a DynamoDB table of employee details and using Amazon Simple Notification
Services (Amazon SNS) to " +
"send employees an email on each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambdaClient.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

Masukkan yang berikut ini di baris perintah untuk menyebarkan fungsi Lambda.

```
node lambda-function-setup.ts
```

Contoh kode ini tersedia [di sini GitHub](#).

Konfigurasi API Gateway untuk menjalankan fungsi Lambda

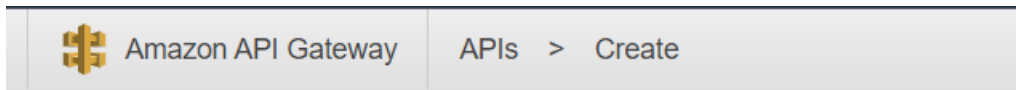
Untuk membangun aplikasi:

1. [Buat API sisanya](#)
2. [Uji metode API Gateway](#)
3. [Menerapkan metode API Gateway](#)

Buat API sisanya

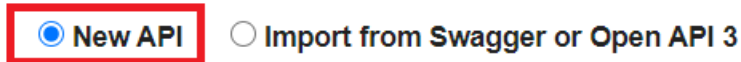
Anda dapat menggunakan konsol API Gateway untuk membuat titik akhir istirahat untuk fungsi Lambda. Setelah selesai, Anda dapat menjalankan fungsi Lambda menggunakan panggilan tenang.

1. Masuk ke [konsol Amazon API Gateway](#).
2. Di bawah Rest API, pilih Build.
3. Pilih API Baru.



Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and meth



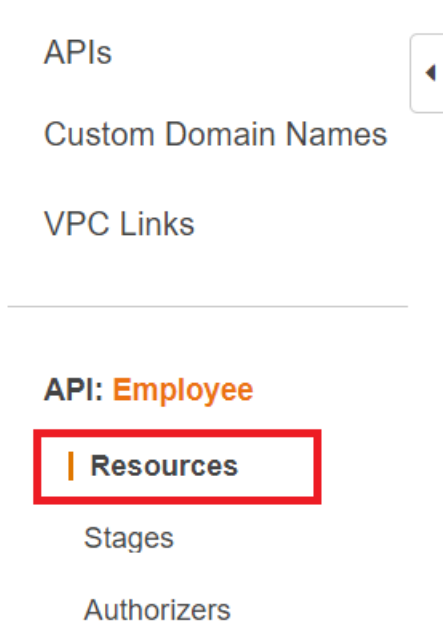
4. Tentukan Karyawan sebagai nama API dan berikan deskripsi.

Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="Employee"/>
Description	<input type="text" value="This invokes a Lambda function"/>
Endpoint Type	<input type="text" value="Regional"/> ⓘ

5. Pilih Buat API.
6. Pilih Sumber Daya di bawah bagian Karyawan.



7. Di bidang nama, tentukan karyawan.

- Pilih Buat sumber daya.
- Dari dropdown Tindakan, pilih Buat Sumber Daya.

Use this page to create a new child resource for your resource. 🟡

Configure as [proxy resource](#) ⓘ

Resource Name*

Resource Path*

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/{proxy+}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/foo`. To handle requests to `/`, add a new ANY method on the `/` resource.

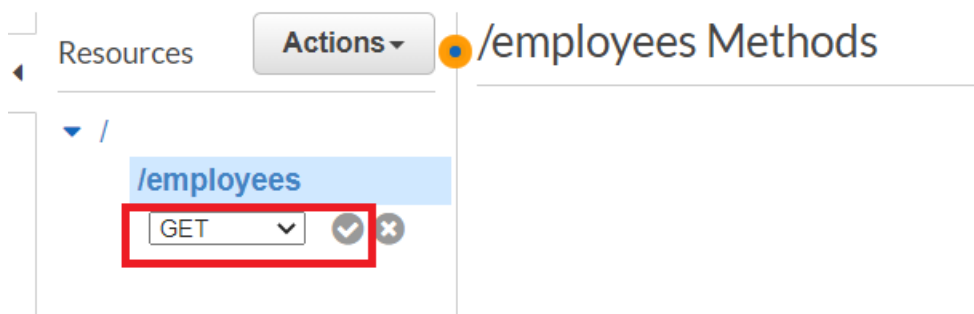
Enable API Gateway CORS ⓘ

* Required

Cancel

Create Resource

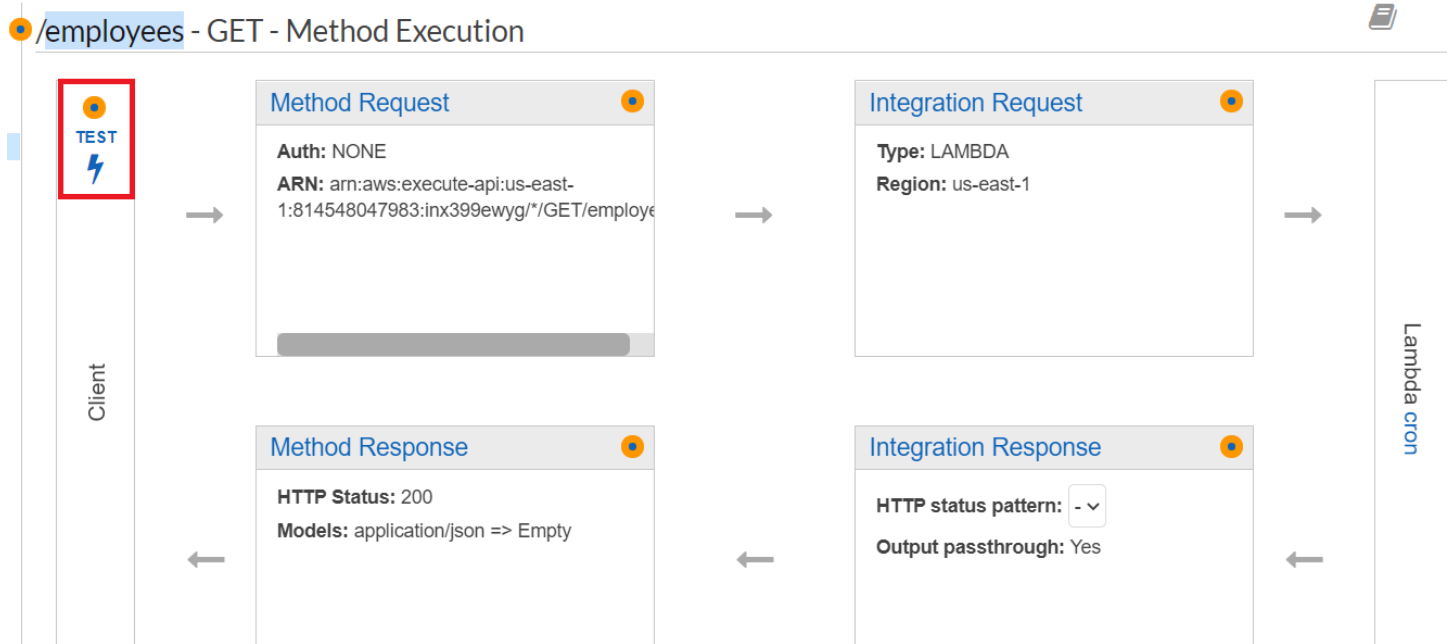
- Pilih `/employees`, pilih Create Method dari Actions, lalu pilih GET dari menu drop-down di bawah `/employees`. Pilih ikon tanda centang.



- Pilih fungsi Lambda dan masukkan `mylambdafunction` sebagai nama fungsi Lambda. Pilih Simpan.

Uji metode API Gateway

Pada titik ini dalam tutorial, Anda dapat menguji metode API Gateway yang memanggil fungsi Lambda `mylambdafunction`. Untuk menguji metode, pilih Uji, seperti yang ditunjukkan pada ilustrasi berikut.

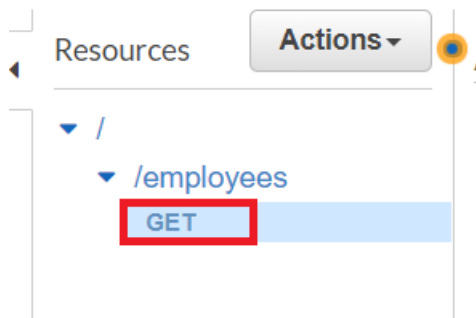


Setelah fungsi Lambda dipanggil, Anda dapat melihat file log untuk melihat pesan yang berhasil.

Menerapkan metode API Gateway

Setelah pengujian berhasil, Anda dapat menerapkan metode dari [konsol Amazon API Gateway](#).

1. Pilih Dapatkan.



2. Dari dropdown Actions, pilih Deploy API.

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	<input type="text" value="[New Stage]"/>
Stage name*	<input type="text" value="lambdastage"/>
Stage description	<input type="text"/>
Deployment description	<input type="text"/>

[Cancel](#) [Deploy](#)

3. Isi formulir Deploy API dan pilih Deploy.

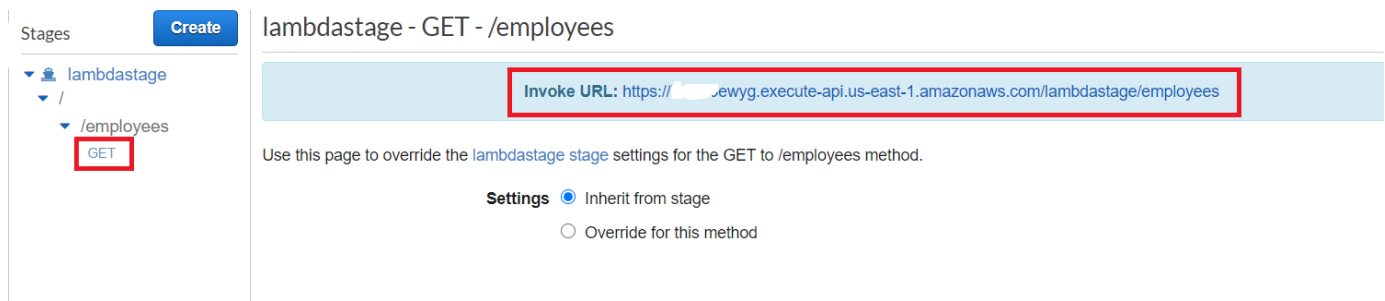
Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	<input type="text" value="[New Stage]"/>
Stage name*	<input type="text" value="lambdastage"/>
Stage description	<input type="text"/>
Deployment description	<input type="text"/>

[Cancel](#) [Deploy](#)

4. Pilih Simpan Perubahan.
5. Pilih Dapatkan lagi dan perhatikan bahwa URL berubah. Ini adalah URL pemanggilan yang dapat Anda gunakan untuk menjalankan fungsi Lambda.



The screenshot shows the AWS Lambda console interface. On the left, a navigation pane shows a tree structure with 'lambdastage' expanded, then '/', and finally '/employees' with a 'GET' method selected. A 'Create' button is visible at the top left. The main content area is titled 'lambdastage - GET - /employees'. A red box highlights the 'Invoke URL' field, which contains the text: 'https://[redacted].execute-api.us-east-1.amazonaws.com/lambdastage/employees'. Below this, there is a note: 'Use this page to override the lambdastage stage settings for the GET to /employees method.' Under the 'Settings' section, the 'Inherit from stage' radio button is selected, and the 'Override for this method' radio button is unselected.

Hapus sumber daya

Selamat! Anda telah memanggil fungsi Lambda melalui Amazon API Gateway menggunakan AWS SDK for JavaScript Seperti yang dinyatakan di awal tutorial ini, pastikan untuk menghentikan semua sumber daya yang Anda buat saat melalui tutorial ini untuk memastikan bahwa Anda tidak dikenakan biaya. Anda dapat melakukan ini dengan menghapus AWS CloudFormation tumpukan yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini, sebagai berikut:

1. Buka [AWS CloudFormation di konsol AWS manajemen](#).
2. Buka halaman Stacks, dan pilih tumpukan.
3. Pilih Hapus.

Membuat alur AWS kerja tanpa server menggunakan AWS SDK for JavaScript

Anda dapat membuat alur kerja AWS tanpa server dengan menggunakan Step Functions SDK for AWS Java dan. AWS Step Functions Setiap langkah alur kerja diimplementasikan menggunakan AWS Lambda fungsi. Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Step Functions adalah layanan orkestrasi tanpa server yang memungkinkan Anda menggabungkan fungsi Lambda dan layanan lainnya untuk membangun aplikasi bisnis penting. AWS

Note

Anda dapat membuat fungsi Lambda dalam berbagai bahasa pemrograman. Untuk tutorial ini, fungsi Lambda diimplementasikan dengan menggunakan Lambda Java API. Untuk informasi selengkapnya tentang Lambda, lihat [Apa itu Lambda](#).

Dalam tutorial ini, Anda membuat alur kerja yang membuat tiket dukungan untuk organisasi. Setiap langkah alur kerja melakukan operasi pada tiket. Tutorial ini menunjukkan cara menggunakan JavaScript untuk memproses data alur kerja. Misalnya, Anda akan mempelajari cara membaca data yang diteruskan ke alur kerja, cara meneruskan data antar langkah, dan cara memanggil AWS layanan dari alur kerja.

Biaya untuk menyelesaikan: AWS Layanan yang termasuk dalam dokumen ini termasuk dalam [Tingkat AWS Gratis](#).

Catatan: Pastikan untuk menghentikan semua sumber daya yang Anda buat saat mengikuti tutorial ini untuk memastikan bahwa Anda tidak lagi dikenakan biaya.

Topik

- [Tugas prasyarat](#)
- [Buat sumber AWS daya](#)
- [Membuat alur kerja](#)
- [Buat fungsi Lambda](#)
- [Tambahkan fungsi Lambda ke alur kerja](#)
- [Jalankan alur kerja Anda dengan menggunakan konsol Step Functions](#)
- [Hapus AWS sumber daya](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Buat sumber AWS daya

Tutorial ini membutuhkan sumber daya berikut.

- Tabel Amazon DynamoDB bernama Case dengan kunci bernama Id.
- Peran IAM bernama `lambda-support` digunakan untuk memanggil fungsi Lambda. Peran ini memiliki kebijakan yang memungkinkannya memanggil layanan Amazon DynamoDB dan Amazon Simple Email Service dari fungsi Lambda.
- Peran IAM bernama `workflow-support` digunakan untuk memanggil alur kerja.
- Bucket Amazon S3 untuk menampung fungsi Lambda.


Anda dapat membuat sumber daya ini secara manual, tetapi kami merekomendasikan penyediaan sumber daya ini menggunakan AWS Cloud Development Kit (AWS CDK) (AWS CDK) seperti yang dijelaskan dalam tutorial ini.

Buat AWS sumber daya menggunakan AWS CloudFormation

AWS CloudFormation memungkinkan Anda untuk membuat dan menyediakan penyebaran AWS infrastruktur yang dapat diprediksi dan berulang kali. Untuk informasi selengkapnya tentang AWS CloudFormation, lihat [AWS CloudFormation Panduan Pengguna](#).

Untuk membuat AWS CloudFormation tumpukan:

1. Instal dan konfigurasi petunjuk AWS CLI berikut di [Panduan AWS CLI Pengguna](#).
2. Buat file bernama `setup.yaml` di direktori root folder proyek Anda, dan salin konten [di sini GitHub](#) ke dalamnya.

 Note

AWS CloudFormationTemplate dibuat menggunakan yang AWS CDK tersedia [di sini GitHub](#). Untuk informasi selengkapnya tentang AWS CDK, lihat [Panduan Developer AWS Cloud Development Kit \(AWS CDK\)](#).

3. Jalankan perintah berikut dari baris perintah, ganti `STACK_NAME` dengan *nama* unik untuk tumpukan.

 Important

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Untuk informasi selengkapnya tentang parameter `create-stack` perintah, lihat [panduan Referensi AWS CLI Perintah](#), dan [Panduan AWS CloudFormation Pengguna](#).

Membuat AWS sumber daya menggunakan Amazon Web Services Management Console;

Untuk membuat sumber daya untuk aplikasi di konsol, ikuti petunjuk di [Panduan AWS CloudFormation Pengguna](#). Gunakan template yang disediakan buat file bernama `setup.yaml`, dan salin konten [di sini GitHub](#).

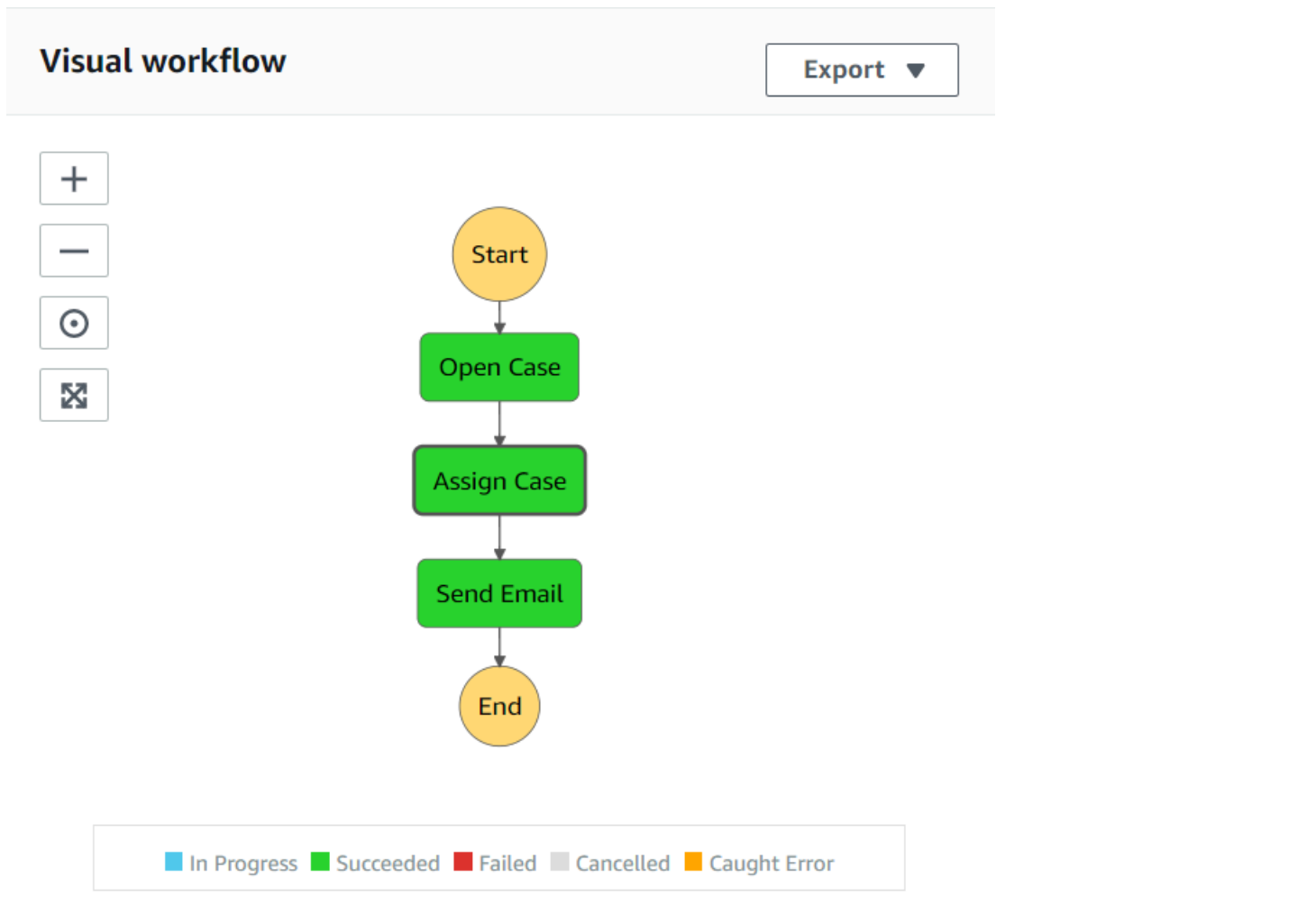
Important

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

Lihat daftar sumber daya di konsol dengan membuka tumpukan di AWS CloudFormation dasbor, dan memilih tab Sumber Daya. Anda membutuhkan ini untuk tutorial.

Membuat alur kerja

Gambar berikut menunjukkan alur kerja yang akan Anda buat dengan tutorial ini.



Berikut ini adalah apa yang terjadi pada setiap langkah dalam alur kerja:

- + Mulai - Memulai alur kerja.
- + Open Case — Menangani nilai ID tiket dukungan dengan meneruskannya ke alur kerja.
- + Tetapkan Kasus - Menetapkan kasus dukungan kepada karyawan dan menyimpan data dalam tabel DynamoDB.
- + Kirim Email - Mengirim pesan email kepada karyawan dengan menggunakan Amazon Simple Email Service (Amazon SES) untuk memberi tahu mereka bahwa ada tiket baru.
- + Akhir - Menghentikan alur kerja.

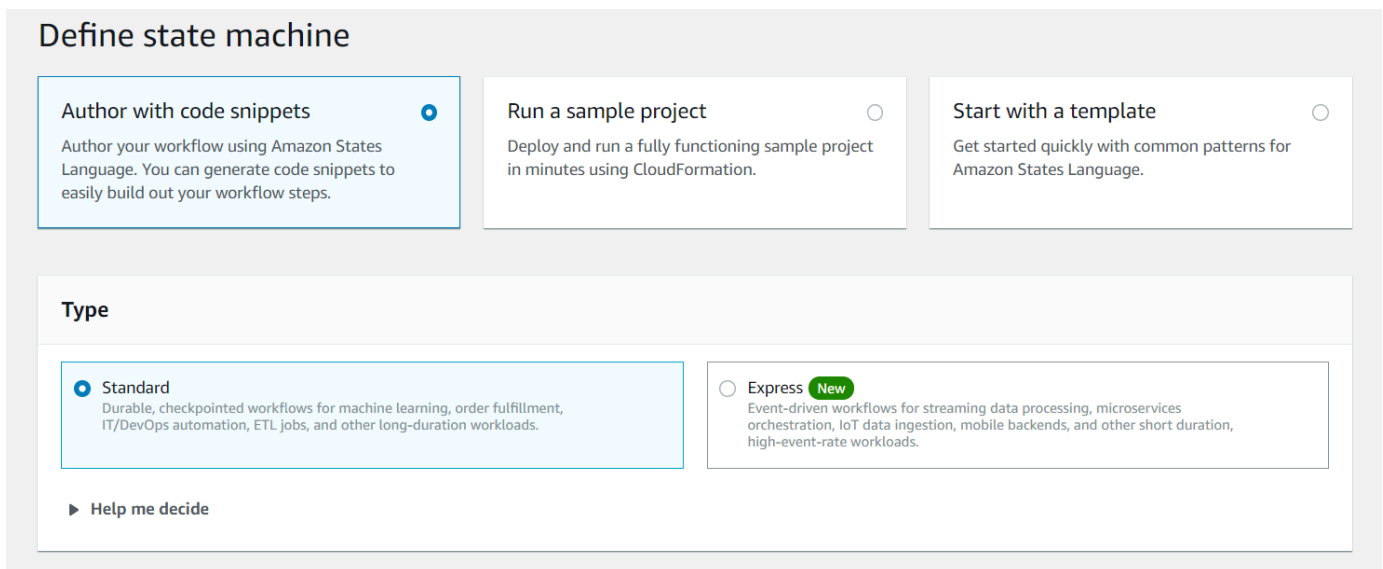
Buat alur kerja tanpa server dengan menggunakan fungsi Langkah

Anda dapat membuat alur kerja yang memproses tiket dukungan. Untuk menentukan alur kerja menggunakan Step Functions, Anda membuat dokumen Amazon States Language (berbasis JSON) untuk menentukan mesin status Anda. Dokumen Amazon States Language menjelaskan setiap langkah. Setelah Anda menentukan dokumen, fungsi Step menyediakan representasi visual dari alur kerja. Gambar berikut menunjukkan dokumen Amazon States Language dan representasi visual dari alur kerja.

Alur kerja dapat meneruskan data antar langkah. Misalnya, langkah Open Case memproses nilai ID kasus (diteruskan ke alur kerja) dan meneruskan nilai tersebut ke langkah Tetapkan Kasus. Kemudian dalam tutorial ini, Anda akan membuat logika aplikasi dalam fungsi Lambda untuk membaca dan memproses nilai data.

Untuk membuat alur kerja

1. Buka [Konsol Amazon Web Services](#).
2. Pilih Buat Mesin Negara.
3. Pilih Penulis dengan cuplikan kode. Di area Type, pilih Standard.



4. Tentukan dokumen Amazon States Language dengan memasukkan kode berikut.

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

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

Note

Jangan khawatir tentang kesalahan yang terkait dengan nilai sumber daya Lambda. Anda akan memperbarui nilai-nilai ini nanti dalam tutorial ini.

5. Pilih Berikutnya.
6. Di bidang nama, masukkan SupportStateMachine.
7. Di bagian Izin, pilih Pilih peran yang ada.
8. Pilih dukungan alur kerja (peran IAM yang Anda buat).

Permissions

Execution role



The IAM role that defines which resources your state machine has permission to access during execution. To create a custom role, go to the [IAM console](#).

Create new role
Let Step Functions create a new role for you based on your state machine's definition and configuration details.

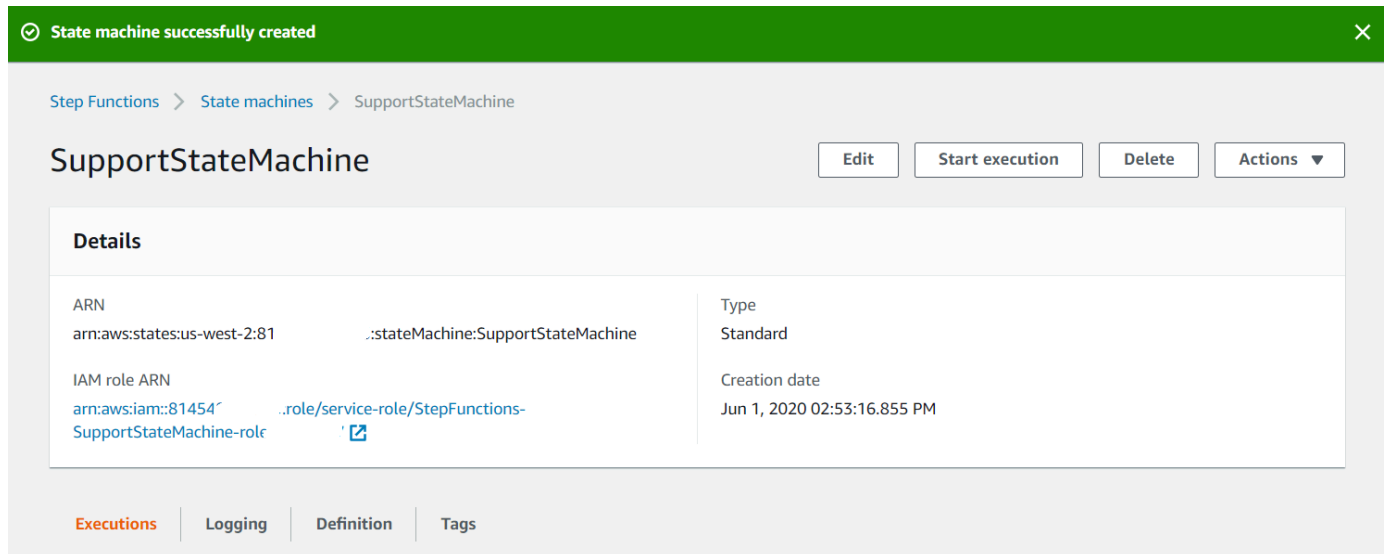
Choose an existing role

Enter a role ARN

Existing roles

workflow-support  

9. Pilih Buat mesin status. Sebuah pesan muncul yang menyatakan mesin negara berhasil dibuat.



Buat fungsi Lambda

Gunakan API runtime Lambda untuk membuat fungsi Lambda. Dalam contoh ini, ada tiga langkah alur kerja yang masing-masing sesuai dengan setiap fungsi Lambda.

Buat fungsi Lambda ini, seperti yang dijelaskan di bagian berikut:

- [Fungsi GetID Lambda](#)- Digunakan sebagai langkah pertama dalam alur kerja yang memproses nilai ID tiket.
- [Kelas Additem Lambda](#)- Digunakan sebagai langkah kedua dalam alur kerja yang menetapkan tiket ke karyawan dan menyimpan data dalam database DynamoDB.
- [sendemail kelas Lambda](#)- Digunakan sebagai langkah ketiga dalam alur kerja yang menggunakan Amazon SES untuk mengirim pesan email kepada karyawan untuk memberi tahu mereka tentang tiket.

Fungsi GetID Lambda

Buat fungsi Lambda yang mengembalikan nilai ID tiket yang diteruskan ke langkah kedua dalam alur kerja.

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

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

Masukkan yang berikut ini di baris perintah untuk menggunakan webpack untuk menggabungkan file ke dalam file bernama `index.js`.

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

Kemudian kompres `index.js` menjadi nama ZIP file `getid.js.zip`. Unggah ZIP file ke bucket Amazon S3 yang Anda buat dalam topik contoh ini.

Contoh kode ini tersedia [di sini GitHub](#).

Kelas Additem Lambda

Buat fungsi Lambda yang memilih karyawan untuk menetapkan tiket, lalu menyimpan data tiket dalam tabel DynamoDB bernama `Case`.

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

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

```

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

Masukkan yang berikut ini di baris perintah untuk menggunakan webpack untuk menggabungkan file ke dalam file bernama `index.js`.

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

Kemudian kompres `index.js` menjadi nama ZIP file `additem.js.zip`. Unggah ZIP file ke bucket Amazon S3 yang Anda buat dalam topik contoh ini.

Contoh kode ini tersedia [di sini GitHub](#).

sendemail kelas Lambda

Buat fungsi Lambda yang mengirim email untuk memberi tahu mereka tentang tiket baru. Alamat email yang diteruskan dari langkah kedua digunakan.

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

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

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

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

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

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

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

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

Masukkan yang berikut ini di baris perintah untuk menggunakan webpack untuk menggabungkan file ke dalam file bernama `index.js`.

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

Kemudian kompres `index.js` menjadi nama ZIP file `sendemail.js.zip`. Unggah ZIP file ke bucket Amazon S3 yang Anda buat dalam topik contoh ini.

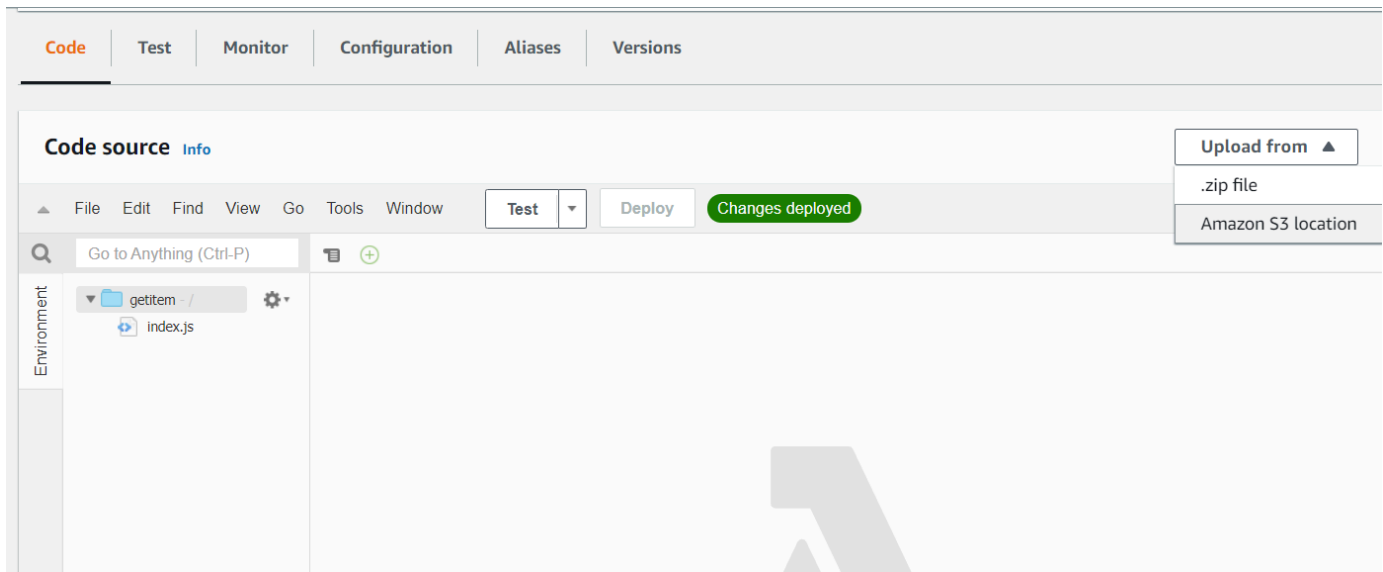
Contoh kode ini tersedia [di sini GitHub](#).

Menyebarkan fungsi Lambda

Untuk menerapkan fungsi `getid` Lambda:

1. Buka konsol Lambda di [Amazon Web Services](#) Console.
2. Pilih Buat Fungsi.
3. Pilih Penulis dari scratch.

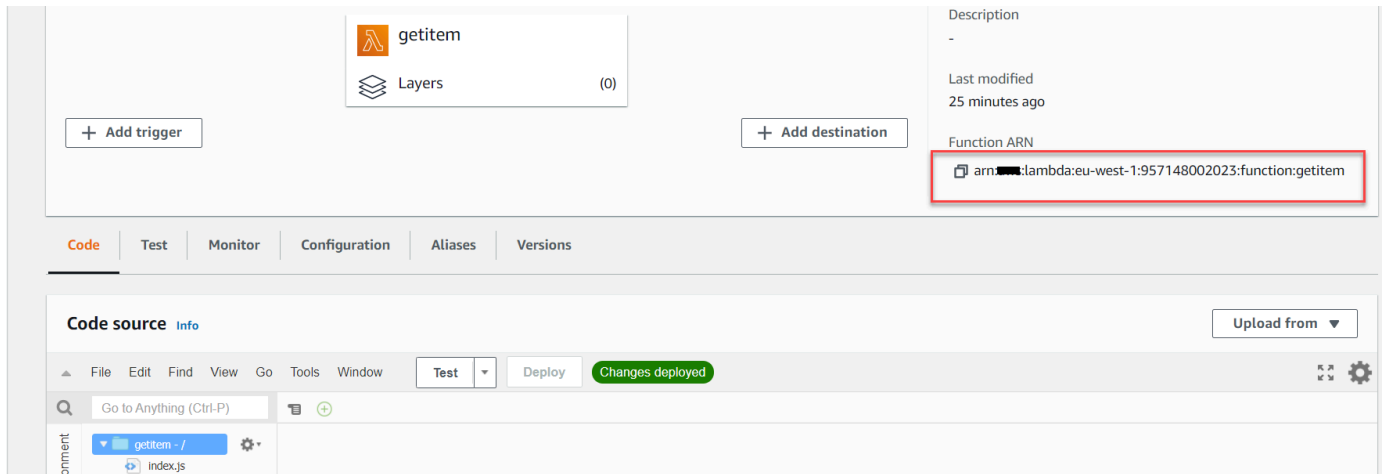
4. Di bagian Informasi dasar, masukkan getid sebagai namanya.
5. Di Runtime, pilih Node.js 14x.
6. Pilih Gunakan peran yang ada, lalu pilih dukungan lambda (peran IAM yang Anda buat di).
7. Pilih Buat fungsi.
8. pilih Unggah dari - Lokasi Amazon S3.
9. Pilih Unggah, pilih Unggah dari - Lokasi Amazon S3, dan masukkan URL tautan Amazon S3.



10. Pilih Simpan.
11. Ulangi prosedur ini untuk additem.js.zip dan sendemail.js.zip ke fungsi Lambda baru. Setelah selesai, Anda akan memiliki tiga fungsi Lambda yang dapat Anda referensikan dalam dokumen Bahasa Negara Amazon.

Tambahkan fungsi Lambda ke alur kerja

1. Buka Konsol Lambda. Perhatikan bahwa Anda dapat melihat nilai Lambda Amazon Resource Name (ARN) di sudut kanan atas.



2. Salin nilainya lalu tempel ke langkah 1 dokumen Amazon States Language, yang terletak di konsol Step Functions.
3. Perbarui Sumber Daya untuk Langkah Tetapkan Kasus dan Kirim Email. Ini adalah bagaimana Anda menghubungkan fungsi Lambda yang dibuat dengan menggunakan AWS SDK for Java ke dalam alur kerja yang dibuat dengan menggunakan Step Functions.

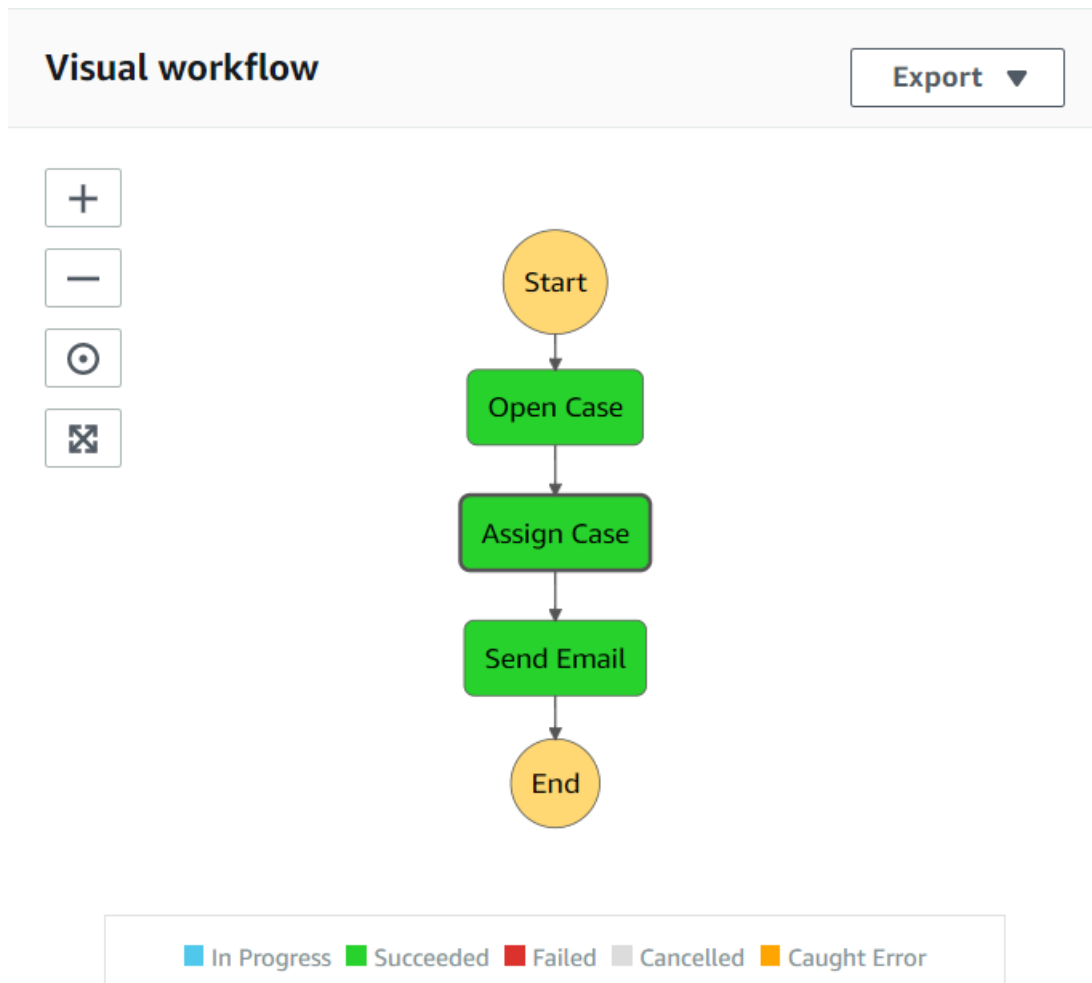
Jalankan alur kerja Anda dengan menggunakan konsol Step Functions

Anda dapat memanggil alur kerja di konsol Step Functions. Eksekusi menerima masukan JSON. Untuk contoh ini, Anda dapat meneruskan data JSON berikut ke alur kerja.

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

Untuk menjalankan alur kerja Anda:

1. Pada konsol Step Functions, pilih Mulai eksekusi.
2. Di bagian Input, lewati data JSON. Lihat alur kerja. Saat setiap langkah selesai, warnanya menjadi hijau.



3. Jika langkah berubah menjadi merah, kesalahan terjadi. Anda dapat mengklik langkah dan melihat log yang dapat diakses dari sisi kanan.

Code | **Step details**

Name Type

Assign Case Task

Status

✔ Succeeded

Resource

arn:aws:lambda:us-west-2:8145-... :function:function3 [↗](#) **CloudWatch**

logs [↗](#)

▶ **Input**

▶ **Output**

▶ **Exception**

Ketika alur kerja selesai, Anda dapat melihat data dalam tabel DynamoDB.

Scan: [Table] Case: id ^

Scan ▾ [Table] Case: id ▾ ^

+ Add filter

Start search

<input type="checkbox"/>	id ⓘ ▲	email ▾	name ▾	registrationDate ▾
<input type="checkbox"/>	001	tblue@noServer.com	Tom Blue	1586217600
<input type="checkbox"/>	091	swhite@noServer.com	Sarah White	1586217600
<input type="checkbox"/>	111	tblue@noServer.com	Tom Blue	1586217600
<input type="checkbox"/>	888	swhite@noServer.com	Sarah White	1586217600

Hapus AWS sumber daya

Selamat, Anda telah membuat alur kerja AWS tanpa server dengan menggunakan SDK for AWS Java. Seperti yang dinyatakan di awal tutorial ini, pastikan untuk menghentikan semua sumber daya yang Anda buat saat melalui tutorial ini untuk memastikan bahwa Anda tidak dikenakan biaya. Anda dapat melakukan ini dengan menghapus AWS CloudFormation tumpukan yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini, sebagai berikut:

1. Buka [AWS CloudFormation di konsol AWS manajemen](#).
2. Buka halaman Stacks, dan pilih tumpukan.
3. Pilih Hapus.

Membuat acara terjadwal untuk menjalankan AWS Lambda fungsi

Anda dapat membuat acara terjadwal yang memanggil AWS Lambda fungsi dengan menggunakan Amazon CloudWatch Event. Anda dapat mengonfigurasi CloudWatch Peristiwa untuk menggunakan ekspresi cron untuk menjadwalkan saat fungsi Lambda dipanggil. Misalnya, Anda dapat menjadwalkan CloudWatch Acara untuk menjalankan fungsi Lambda setiap hari kerja.

AWS Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Anda dapat membuat fungsi Lambda dalam berbagai bahasa pemrograman. Untuk informasi selengkapnya tentang AWS Lambda, lihat [Apa itu AWS Lambda](#).

Dalam tutorial ini, Anda membuat fungsi Lambda dengan menggunakan JavaScript Lambda runtime API. Contoh ini menginvokasi layanan AWS yang berbeda untuk melakukan kasus penggunaan tertentu. Misalnya, asumsikan bahwa organisasi mengirim pesan teks seluler kepada karyawannya yang memberi selamat kepada mereka pada tanggal peringatan satu tahun, seperti yang ditunjukkan dalam ilustrasi ini.

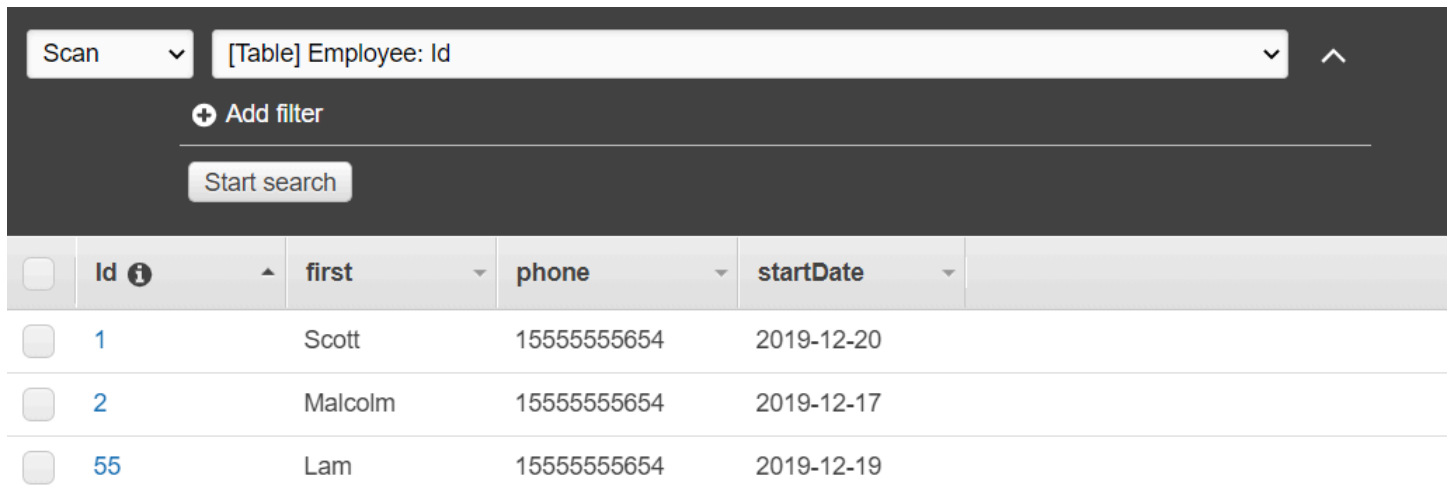


Tutorial akan memakan waktu sekitar 20 menit untuk menyelesaikannya.

Tutorial ini menunjukkan cara menggunakan JavaScript logika untuk membuat solusi yang melakukan kasus penggunaan ini. Misalnya, Anda akan mempelajari cara membaca database untuk menentukan karyawan mana yang telah mencapai tanggal ulang tahun satu tahun, cara memproses data, dan mengirim pesan teks semuanya dengan menggunakan fungsi Lambda. Kemudian Anda akan belajar cara menggunakan ekspresi cron untuk menjalankan fungsi Lambda setiap hari kerja.

AWSTutorial ini menggunakan tabel Amazon DynamoDB bernama Employee yang berisi bidang-bidang ini.

- id - kunci utama untuk tabel.
- FirstName - nama depan karyawan.
- telepon - nomor telepon karyawan.
- StartDate - tanggal mulai karyawan.



The screenshot shows a table viewer interface for a table named 'Employee'. The table has columns for 'Id', 'first', 'phone', and 'startDate'. There are three rows of data displayed.

<input type="checkbox"/>	Id <i>i</i>	first	phone	startDate
<input type="checkbox"/>	1	Scott	15555555654	2019-12-20
<input type="checkbox"/>	2	Malcolm	15555555654	2019-12-17
<input type="checkbox"/>	55	Lam	15555555654	2019-12-19

Important

Biaya untuk menyelesaikan: AWS Layanan yang termasuk dalam dokumen ini termasuk dalam Tingkat AWS Gratis. Namun, pastikan untuk menghentikan semua sumber daya setelah Anda menyelesaikan tutorial ini untuk memastikan bahwa Anda tidak dikenakan biaya.

Untuk membangun aplikasi:

1. [Prasyarat lengkap](#)

2. [Buat sumber AWS daya](#)
3. [Siapkan skrip browser](#)
4. [Buat dan unggah fungsi Lambda](#)
5. [Menyebarkan fungsi Lambda](#)
6. [Jalankan aplikasi](#)
7. [Hapus sumber daya](#)

Tugas prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node.js ini, dan instal modul wajib AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Buat sumber AWS daya

Tutorial ini membutuhkan sumber daya berikut.

- Tabel Amazon DynamoDB bernama Employee dengan kunci bernama Id dan bidang yang ditunjukkan pada ilustrasi sebelumnya. Pastikan Anda memasukkan data yang benar, termasuk ponsel yang valid yang ingin Anda uji kasus penggunaan ini. Untuk informasi selengkapnya, lihat [Membuat Tabel](#).
- Peran IAM dengan izin terlampir untuk menjalankan fungsi Lambda.
- Bucket Amazon S3 untuk menampung fungsi Lambda.

Anda dapat membuat sumber daya ini secara manual, tetapi kami merekomendasikan penyediaan sumber daya ini menggunakan AWS CloudFormation seperti yang dijelaskan dalam tutorial ini.

Buat sumber AWS daya menggunakan AWS CloudFormation

AWS CloudFormation memungkinkan Anda untuk membuat dan menyediakan penyebaran AWS infrastruktur yang dapat diprediksi dan berulang kali. Untuk informasi selengkapnya tentang AWS CloudFormation, lihat [Panduan Pengguna AWS CloudFormation](#).

Untuk membuat AWS CloudFormation tumpukan menggunakan AWS CLI:

1. Instal dan konfigurasi petunjuk AWS CLI berikut di [Panduan AWS CLI Pengguna](#).
2. Buat file bernama `setup.yaml` di direktori root folder proyek Anda, dan salin konten [di sini GitHub](#) ke dalamnya.

Note

AWS CloudFormationTemplate dibuat menggunakan yang AWS CDK tersedia [di sini GitHub](#). Untuk informasi selengkapnya tentang AWS CDK, lihat [Panduan Developer AWS Cloud Development Kit \(AWS CDK\)](#).

3. Jalankan perintah berikut dari baris perintah, ganti *STACK_NAME* dengan *nama* unik untuk tumpukan.

Important

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Untuk informasi selengkapnya tentang parameter `create-stack` perintah, lihat [panduan Referensi AWS CLI Perintah](#), dan [Panduan AWS CloudFormation Pengguna](#).

Lihat daftar sumber daya di konsol dengan membuka tumpukan di AWS CloudFormation dasbor, dan memilih tab Sumber Daya. Anda membutuhkan ini untuk tutorial.

4. Saat tumpukan dibuat, gunakan AWS SDK for JavaScript untuk mengisi tabel DynamoDB, seperti yang dijelaskan dalam [Mengisi tabel DynamoDB](#)

Mengisi tabel DynamoDB

Untuk mengisi tabel, pertama membuat direktori bernama `libs`, dan di dalamnya membuat file bernama `dynamoClient.js`, dan paste konten di bawah ini ke dalamnya.

```
const { DynamoDBClient } = require( "@aws-sdk/client-dynamodb" );
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon DynamoDB service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

Kode ini tersedia [di sini GitHub](#).

Selanjutnya, buat file bernama `populate-table.js` di direktori root folder proyek Anda, dan salin konten [di sini GitHub](#) ke dalamnya. Untuk salah satu item, ganti nilai `phone` properti dengan nomor ponsel yang valid dalam format E.164, dan nilai untuk `startDate` dengan tanggal hari ini.

Jalankan perintah berikut dari baris perintah.

```
node populate-table.js
```

```
const {
BatchWriteItemCommand } = require( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require( "../libs/dynamoClient" );
// Set the parameters.
const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
    ],
  },
  PutRequest: {
```

```
    Item: {
      id: { N: "2" },
      firstName: { S: "Xing" },
      phone: { N: "155555555555653" },
      startDate: { S: "2019-12-17" },
    },
  },
},
{
  PutRequest: {
    Item: {
      id: { N: "55" },
      firstName: { S: "Harriette" },
      phone: { N: "155555555555652" },
      startDate: { S: "2019-12-19" },
    },
  },
},
],
},
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Kode ini tersedia [di sini GitHub](#).

Membuat AWS Lambda fungsi

Mengkonfigurasi SDK

Pertama impor modul dan perintah yang diperlukan AWS SDK for JavaScript (v3): `DynamoDBClient` dan `ScanCommand` `DynamoDB`, dan dan perintah `AmazonSNSClient` `SNS.PublishCommand`. Ganti **REGION** dengan AWS Region. Kemudian hitung tanggal hari ini dan tetapkan ke parameter. Kemudian buat parameter untuk `ScanCommand`. Replace **TABLE_NAME** dengan nama tabel yang Anda buat di [Buat sumber AWS daya](#) bagian contoh ini.

Cuplikan kode berikut menunjukkan langkah ini. (Lihat [Bundling fungsi Lambda](#) contoh lengkapnya.)

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

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

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
  want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};
```

Memindai tabel DynamoDB

Pertama buat fungsi async/await yang dipanggil `sendText` untuk mempublikasikan pesan teks menggunakan Amazon SNS. `PublishCommand` Kemudian, tambahkan pola `try` blok yang memindai tabel DynamoDB untuk karyawan dengan ulang tahun kerja mereka hari ini, dan kemudian memanggil fungsi untuk mengirim pesan `sendText` teks kepada karyawan ini. Jika terjadi kesalahan, `catch` blok dipanggil.

Cuplikan kode berikut menunjukkan langkah ini. (Lihat [Bundling fungsi Lambda](#) contoh lengkapnya.)

```
exports.handler = async (event, context, callback) => {
```

```
// Helper function to send message using Amazon SNS.
async function sendText(textParams) {
  try {
    const data = await snsclient.send(new PublishCommand(textParams));
    console.log("Message sent");
  } catch (err) {
    console.log("Error, message not sent ", err);
  }
}
try {
  // Scan the table to check identify employees with work anniversary today.
  const data = await dbclient.send(new ScanCommand(params));
  data.Items.forEach(function (element, index, array) {
    const textParams = {
      PhoneNumber: element.phone.N,
      Message:
        "Hi " +
        element.firstName.S +
        "; congratulations on your work anniversary!",
    };
    // Send message using Amazon SNS.
    sendText(textParams);
  });
} catch (err) {
  console.log("Error, could not scan table ", err);
}
};
```

Bundling fungsi Lambda

Topik ini menjelaskan cara menggabungkan `mylambdafunction.js` dan AWS SDK for JavaScript modul yang diperlukan untuk contoh ini ke dalam file yang dibundel yang disebut `index.js`.

1. Jika Anda belum melakukannya, ikuti contoh ini [Tugas prasyarat](#) untuk menginstal webpack.

Note

Untuk informasi tentang webpack, lihat [Bundel aplikasi dengan webpack](#).

2. Jalankan yang berikut ini di baris perintah untuk menggabungkan JavaScript untuk contoh ini ke dalam file bernama `<index.js>`:

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

Important

Perhatikan output diberi nama `index.js`. Ini karena fungsi Lambda harus memiliki `index.js` handler untuk bekerja.

3. Kompres file output yang dibundel, `index.js`, ke dalam file ZIP bernama `my-lambda-function.zip`.
4. Unggah `mylambdafunction.zip` ke bucket Amazon S3 yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini.

Berikut adalah kode skrip browser lengkap untuk `mylambdafunction.js`.

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

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

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

```
// Set the projection expression, which the the attributes that you want.
ProjectionExpression: "firstName, phone",
TableName: "TABLE_NAME",
};

// Create the client service objects.
const dbclient = new DynamoDBClient({ region: REGION });
const snsclient = new SNSClient({ region: REGION });

exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!";
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Deploy fungsi Lambda

Di root proyek Anda, buat `lambda-function-setup.js` file, dan tempel konten di bawah ini ke dalamnya.

Ganti *BUCKET_NAME* dengan nama bucket Amazon S3 tempat Anda mengunggah versi ZIP fungsi Lambda Anda. Ganti *ZIP_FILE_NAME* dengan *nama nama* versi ZIP dari fungsi Lambda Anda. Ganti *IAM_ROLE_ARN* dengan Amazon Resource Number (ARN) dari peran IAM yang Anda buat dalam topik tutorial ini. [Buat sumber AWS daya](#) Ganti *LAMBDA_FUNCTION_NAME* dengan *nama* untuk fungsi Lambda.

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand,
} = require("@aws-sdk/client-lambda");
const {
  lambdaClient
} = require("../libs/lambdaClient.js");

// Instantiate an Lambda client service object.
const lambda = new LambdaClient({ region: REGION });

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
  lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
  Services (Amazon SNS) to " +
    "send employees an email the each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambda.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
```

```
run();
```

Masukkan yang berikut ini di baris perintah untuk menyebarkan fungsi Lambda.

```
node lambda-function-setup.js
```

Contoh kode ini tersedia [di sini GitHub](#).

Konfigurasi CloudWatch untuk menjalankan fungsi Lambda

Untuk mengkonfigurasi CloudWatch untuk menjalankan fungsi Lambda:

1. Buka halaman Fungsi di konsol Lambda.
2. Pilih fungsi Lambda.
3. Di bawah Desainer, pilih Tambahkan pemicu.
4. Atur tipe pemicu ke CloudWatch EventBridgeEvents/.
5. Untuk Aturan, pilih Buat aturan baru.
6. Isi nama aturan dan deskripsi aturan.
7. Untuk jenis aturan, pilih Ekspresi jadwal.
8. Di bidang Ekspresi Jadwal, masukkan ekspresi cron. Misalnya, cron (0 12? * SEN-JUMAT *).
9. Pilih Tambahkan.

Note

Untuk informasi selengkapnya, lihat [Menggunakan Lambda dengan CloudWatch Acara](#).

Hapus sumber daya

Selamat! Anda telah memanggil fungsi Lambda melalui acara terjadwal CloudWatch Amazon menggunakan AWS SDK for JavaScript Seperti yang dinyatakan di awal tutorial ini, pastikan untuk menghentikan semua sumber daya yang Anda buat saat melalui tutorial ini untuk memastikan bahwa Anda tidak dikenakan biaya. Anda dapat melakukan ini dengan menghapus AWS CloudFormation tumpukan yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini, sebagai berikut:

1. Buka [konsol AWS CloudFormation](#).
2. Pada halaman Stacks, pilih tumpukan.

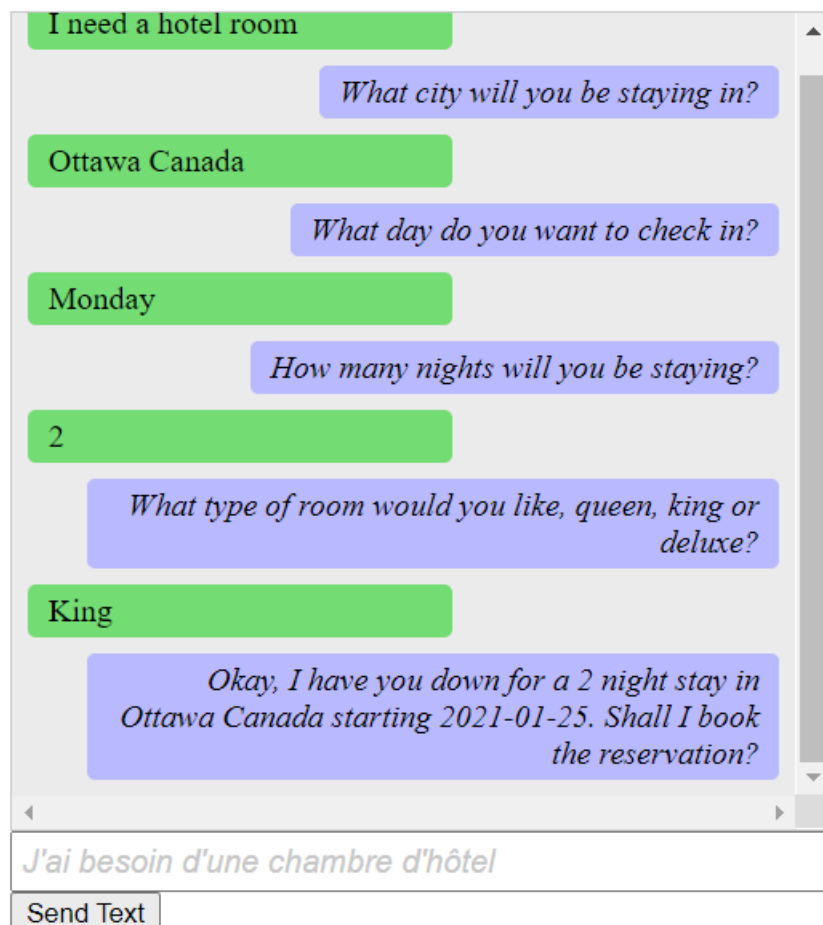
3. Pilih Hapus.

Membangun chatbot Amazon Lex

Anda dapat membuat chatbot Amazon Lex dalam aplikasi web untuk melibatkan pengunjung situs web Anda. Chatbot Amazon Lex adalah fungsi yang melakukan percakapan obrolan online dengan pengguna tanpa memberikan kontak langsung dengan seseorang. Misalnya, ilustrasi berikut menunjukkan chatbot Amazon Lex yang melibatkan pengguna tentang pemesanan kamar hotel.

Amazon Lex - BookTrip

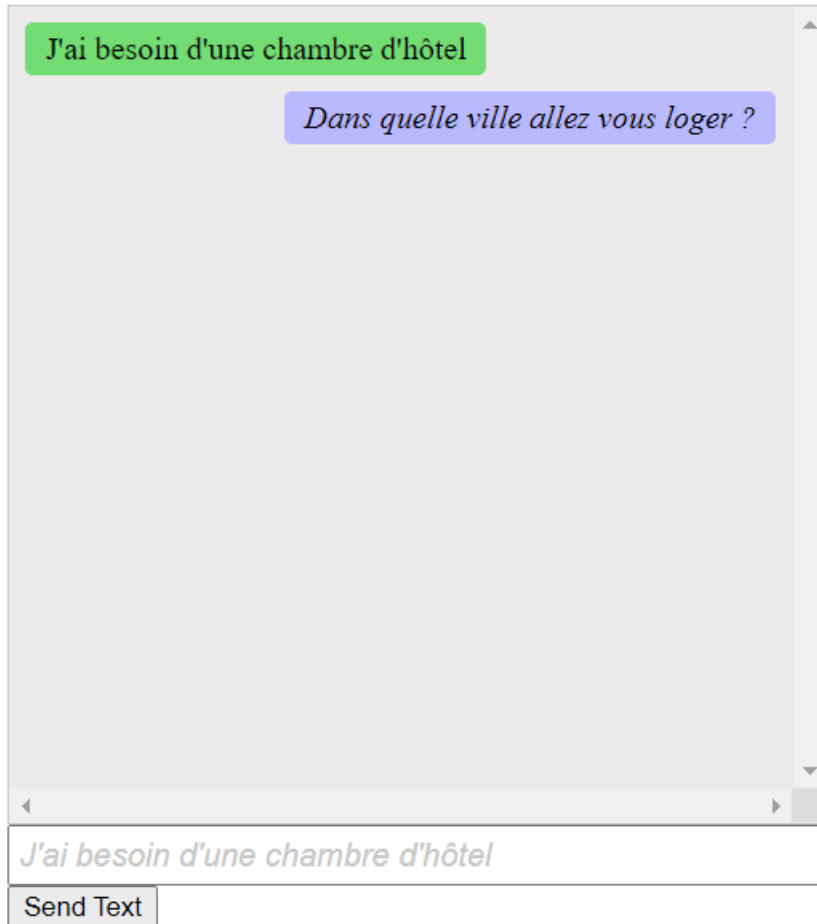
This multiple language chatbot shows you how easy it is to incorporate [Amazon Lex](#) into your web apps. Try it out.



Chatbot Amazon Lex yang dibuat dalam AWS tutorial ini mampu menangani beberapa bahasa. Misalnya, pengguna yang berbicara bahasa Prancis dapat memasukkan teks Prancis dan mendapatkan kembali tanggapan dalam bahasa Prancis.

Amazon Lex - BookTrip

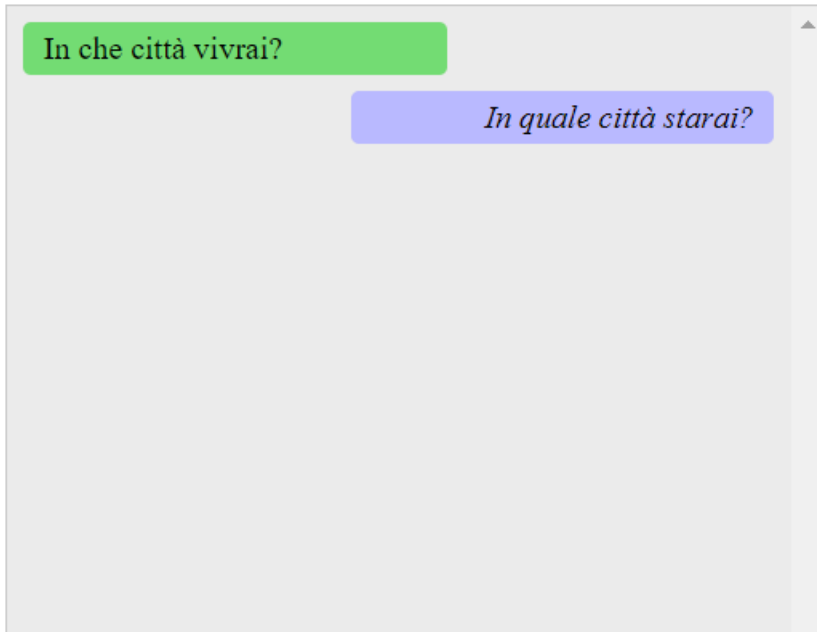
This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



Demikian juga, pengguna dapat berkomunikasi dengan chatbot Amazon Lex dalam bahasa Italia.

Amazon Lex - BookTrip

This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



AWSTutorial ini memandu Anda melalui pembuatan chatbot Amazon Lex dan mengintegrasikannya ke dalam aplikasi web Node.js. AWS SDK for JavaScript(v3) digunakan untuk memanggil layanan iniAWS:

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

Biaya untuk menyelesaikan: AWS Layanan yang termasuk dalam dokumen ini termasuk dalam [Tingkat AWS Gratis](#).

Catatan: Pastikan untuk menghentikan semua sumber daya yang Anda buat saat mengikuti tutorial ini untuk memastikan bahwa Anda tidak dikenakan biaya.

Untuk membangun aplikasi:

1. [Prasyarat](#)
2. [Sumber daya penyediaan](#)

3. [Buat chatbot Amazon Lex](#)
4. [Buat HTML](#)
5. [Buat skrip browser](#)
6. [Langkah selanjutnya](#)

Prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#). Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Buat sumber AWS daya

Tutorial ini membutuhkan sumber daya berikut.

- Peran IAM yang tidak diautentikasi dengan izin terlampir untuk:
 - Amazon Comprehend
 - Amazon Translate
 - Amazon Lex

Anda dapat membuat sumber daya ini secara manual, tetapi kami merekomendasikan penyediaan sumber daya ini menggunakan AWS CloudFormation seperti yang dijelaskan dalam tutorial ini.

Buat sumber AWS daya menggunakan AWS CloudFormation

AWS CloudFormation memungkinkan Anda untuk membuat dan menyediakan penyebaran AWS infrastruktur yang dapat diprediksi dan berulang kali. Untuk informasi selengkapnya tentang AWS CloudFormation, lihat [Panduan Pengguna AWS CloudFormation](#).

Untuk membuat AWS CloudFormation tumpukan menggunakan AWS CLI:

1. Instal dan konfigurasi petunjuk AWS CLI berikut di [Panduan AWS CLI Pengguna](#).
2. Buat file bernama `setup.yaml` di direktori root folder proyek Anda, dan salin konten [di sini GitHub](#) ke dalamnya.

Note

AWS CloudFormationTemplate dibuat menggunakan yang AWS CDK tersedia [di sini GitHub](#). Untuk informasi selengkapnya tentang AWS CDK, lihat [Panduan Developer AWS Cloud Development Kit \(AWS CDK\)](#).

3. Jalankan perintah berikut dari baris perintah, ganti *STACK_NAME* dengan *nama* unik untuk tumpukan.

Important

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Untuk informasi selengkapnya tentang parameter `create-stack` perintah, lihat [panduan Referensi AWS CLI Perintah](#), dan [Panduan AWS CloudFormation Pengguna](#).

Untuk melihat sumber daya yang dibuat, buka konsol Amazon Lex, pilih tumpukan, dan pilih tab Sumber Daya.

Buat bot Amazon Lex

⚠ Important

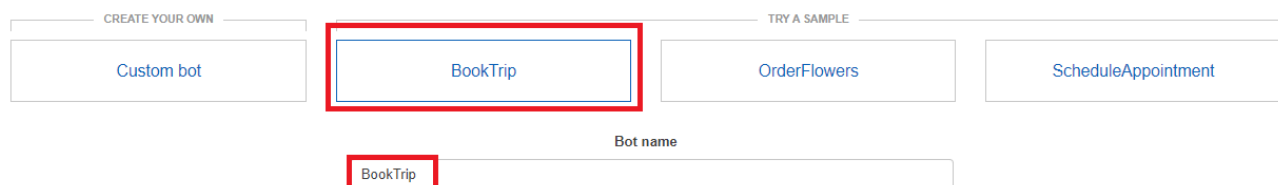
Gunakan V1 dari konsol Amazon Lex untuk membuat bot. Contoh ini tidak berfungsi dengan bot yang dibuat menggunakan V2.

Langkah pertama adalah membuat chatbot Amazon Lex dengan menggunakan Amazon Web Services Management Console. Dalam contoh ini, contoh Amazon Lex BookTrip digunakan. Untuk informasi selengkapnya, lihat [Pesan Perjalanan](#).

- Masuk ke Amazon Web Services Management Console dan buka konsol Amazon Lex di [Amazon Web Services Console](#).
- Pada halaman Bots, pilih Buat.
- Pilih BookTrip cetak biru (tinggalkan nama bot default). BookTrip

Create your bot

Amazon Lex enables any developer to build conversational chatbots quickly and easily. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversational flow directly from the console, and then Amazon Lex manages the dialogue and dynamically adjusts the response. To get started, you can choose one of the sample bots provided below or build a new custom bot from scratch.



- Isi pengaturan default dan pilih Buat (konsol menunjukkan BookTripbot). Pada tab Editor, tinjau detail maksud yang telah dikonfigurasi sebelumnya.
- Uji bot di jendela uji. Mulai tes dengan mengetik Saya ingin memesan kamar hotel.

> Test bot (Latest)

✔ Ready. Build complete

I want to book a hotel room

What city will you be staying in?

Clear chat history

 Chat with your bot...

Inspect response

Dialog State: ElicitSlot

[Hide](#)

Summary Detail

Intent: BookHotel

- Pilih Publikasikan dan tentukan nama alias (Anda akan memerlukan nilai ini saat menggunakan AWS SDK for JavaScript).

Note

Anda perlu mereferensikan nama bot dan alias bot dalam JavaScript kode Anda.

Buat HTML

Buat file bernama `index.html`. Salin dan tempel kode di bawah ini ke `index.html`. Referensi HTML `inimain.js`. Ini adalah versi bundel dari `index.js`, yang mencakup AWS SDK for JavaScript modul yang diperlukan. Anda akan membuat file ini di [Buat HTML](#). `index.html` juga referensi `style.css`, yang menambahkan gaya.

```
<!doctype html>
<head>
  <title>Amazon Lex - Sample Application (BookTrip)</title>
```

```
<link type="text/css" rel="stylesheet" href="style.css" />
</head>

<body>
  <h1 id="title">Amazon Lex - BookTrip</h1>
  <p id="intro">
    This multiple language chatbot shows you how easy it is to incorporate
    <a
      href="https://aws.amazon.com/lex/"
      title="Amazon Lex (product)"
      target="_new"
    >Amazon Lex</a>
    >
    into your web apps. Try it out.
  </p>
  <div id="conversation"></div>
  <input
    type="text"
    id="wisdom"
    size="80"
    value=""
    placeholder="J'ai besoin d'une chambre d'hôtel"
  />
  <br />
  <button onclick="createResponse()">Send Text</button>
  <script type="text/javascript" src="./main.js"></script>
</body>
```

Kode ini juga tersedia [di sini GitHub](#).

Buat skrip browser

Buat file bernama `index.js`. Salin dan tempel kode di bawah ini ke dalam `index.js`. Impor AWS SDK for JavaScript modul dan perintah yang diperlukan. Buat klien untuk Amazon Lex, Amazon Comprehend, dan Amazon Translate. Ganti **AWSREGION** dengan Region, dan **IDENTITY_POOL_ID** dengan ID kumpulan identitas yang Anda buat di [Buat sumber AWS daya](#). Untuk mengambil ID kumpulan identitas ini, buka kumpulan identitas di konsol Amazon Cognito, pilih Edit kumpulan identitas, dan pilih Contoh kode di menu samping. ID kumpulan identitas ditampilkan dalam teks merah di konsol.

Pertama, buat `libs` direktori buat objek klien layanan yang diperlukan dengan membuat tiga file, `comprehendClient.js`, `lexClient.js`, dan `translateClient.js`. Rekatkan kode yang sesuai di bawah ini ke masing-masing, dan ganti `REGION` dan `IDENTITY_POOL_ID` di setiap file.

Note

Gunakan ID kumpulan identitas Amazon Cognito yang Anda buat. [Buat sumber AWS daya menggunakan AWS CloudFormation](#)

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

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

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

export { comprehendClient };
```

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

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

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

```
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }},
});

export { lexClient };
```

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

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

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

export { translateClient };
```

Kode ini tersedia [di sini GitHub](#) .

Selanjutnya, buat `index.js` file, dan tempel kode di bawah ini ke dalamnya.

Ganti `BOT_ALIAS` dan `BOT_NAME` dengan `alias` dan `nama bot Amazon Lex Anda masing-masing`, dan `USER_ID` dengan `id pengguna`. Fungsi `createResponse` asinkron melakukan hal berikut:

- Mengambil teks yang dimasukkan oleh pengguna ke browser dan menggunakan Amazon Comprehend untuk menentukan kode bahasanya.
- Mengambil kode bahasa dan menggunakan Amazon Translate untuk menerjemahkan teks ke dalam bahasa Inggris.
- Mengambil teks yang diterjemahkan dan menggunakan Amazon Lex untuk menghasilkan respons.
- Memposting respons ke halaman browser.

```
import { DetectDominantLanguageCommand } from "@aws-sdk/client-comprehend";
import { TranslateTextCommand } from "@aws-sdk/client-translate";
import { PostTextCommand } from "@aws-sdk/client-lex-runtime-service";
import { lexClient } from "../libs/lexClient.js";
import { translateClient } from "../libs/translateClient.js";
import { comprehendClient } from "../libs/comprehendClient.js";

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

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

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

  var lexTextResponse = lexResponse;

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

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

```
function loadNewItem() {
  showRequest();

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

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


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

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

Kode ini tersedia [di sini GitHub](#).

Sekarang gunakan webpack untuk menggabungkan AWS SDK for JavaScript modul `index.js` dan ke dalam satu file, `main.js`.

1. Jika Anda belum melakukannya, ikuti contoh ini [Prasyarat](#) untuk menginstal webpack.

 Note

Untuk informasi tentang webpack, lihat [Bundel aplikasi dengan webpack](#).

2. Jalankan yang berikut ini di baris perintah untuk menggabungkan JavaScript untuk contoh ini ke dalam file bernama `main.js`:

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

Langkah selanjutnya

Selamat! Anda telah membuat aplikasi Node.js yang menggunakan Amazon Lex untuk menciptakan pengalaman pengguna yang interaktif. Seperti yang dinyatakan di awal tutorial ini, pastikan untuk menghentikan semua sumber daya yang Anda buat saat melalui tutorial ini untuk memastikan bahwa Anda tidak dikenakan biaya. Anda dapat melakukan ini dengan menghapus AWS CloudFormation tumpukan yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini, sebagai berikut:

1. Buka [konsol AWS CloudFormation](#).
2. Pada halaman Stacks, pilih tumpukan.
3. Pilih Hapus.

Untuk contoh AWS lintas layanan lainnya, lihat contoh [AWS SDK for JavaScript lintas layanan](#).

Membuat contoh aplikasi perpesanan

Anda dapat membuat AWS aplikasi yang mengirim dan mengambil pesan dengan menggunakan Amazon Simple Queue Service (Amazon SQS) AWS SDK for JavaScript dan Amazon Simple Queue Service (Amazon SQS). Pesan disimpan dalam antrean masuk pertama, keluar pertama (FIFO) yang memastikan bahwa urutan pesan konsisten. Misalnya, pesan pertama yang disimpan dalam antrian adalah pesan pertama yang dibaca dari antrian.

Note

Untuk informasi selengkapnya tentang Amazon SQS, lihat [Apa itu Layanan Antrian Sederhana Amazon?](#)

Dalam tutorial ini, Anda membuat aplikasi Node.js bernama AWS Messaging.

Biaya untuk menyelesaikan: AWS Layanan yang termasuk dalam dokumen ini termasuk dalam [Tingkat AWS Gratis](#).

Catatan: Pastikan untuk menghentikan semua sumber daya yang Anda buat saat mengikuti tutorial ini untuk memastikan bahwa Anda tidak dikenakan biaya.

Untuk membangun aplikasi:

1. [Prasyarat](#)

2. [Sumber daya penyedia](#)
3. [Memahami alur kerja](#)
4. [Buat HTML](#)
5. [Buat skrip browser](#)
6. [Langkah selanjutnya](#)

Prasyarat

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

- Siapkan lingkungan proyek untuk menjalankan TypeScript contoh Node ini, dan instal modul yang diperlukan AWS SDK for JavaScript dan pihak ketiga. Ikuti instruksi pada [GitHub](#).
- Buat file konfigurasi bersama dengan kredensi pengguna Anda. Untuk informasi selengkapnya tentang menyediakan file kredensial bersama, lihat File [konfigurasi dan kredensial bersama](#) di Panduan Referensi AWSSDK dan Alat.

Important

Contoh ini menggunakan ECMAScript6 (ES6). Ini membutuhkan Node.js versi 13.x atau lebih tinggi. Untuk mengunduh dan menginstal versi terbaru dari Node.js, lihat [unduh Node.js](#). Namun, jika Anda lebih suka menggunakan sintaks CommonJS, silakan merujuk ke [JavaScript ES6/CommonJS sintaks](#)

Buat sumber AWS daya

Tutorial ini membutuhkan sumber daya berikut.

- Peran IAM yang tidak diautentikasi dengan izin untuk Amazon SQS.
- [Antrian Amazon SQS FIFO bernama Message.FIFO - untuk informasi tentang membuat antrian, lihat Membuat antrian Amazon SQS.](#)

Anda dapat membuat sumber daya ini secara manual, tetapi kami merekomendasikan penyediaan sumber daya ini menggunakan AWS CloudFormation (AWS CloudFormation) seperti yang dijelaskan dalam tutorial ini.

Note

AWS CloudFormation ini adalah kerangka pengembangan perangkat lunak yang memungkinkan Anda untuk menentukan sumber daya aplikasi cloud. Untuk informasi selengkapnya, silakan lihat [Panduan Pengguna AWS CloudFormation](#).

Buat AWS sumber daya menggunakan AWS CloudFormation

AWS CloudFormation memungkinkan Anda untuk membuat dan menyediakan penyebaran AWS infrastruktur yang dapat diprediksi dan berulang kali. Untuk informasi selengkapnya tentang AWS CloudFormation, lihat [Panduan Pengguna AWS CloudFormation](#).

Untuk membuat AWS CloudFormation tumpukan menggunakan AWS CLI:

1. Instal dan konfigurasi petunjuk AWS CLI berikut di [Panduan AWS CLI Pengguna](#).
2. Buat file bernama `setup.yaml` di direktori root folder proyek Anda, dan salin konten [di sini GitHub](#) ke dalamnya.

Note

AWS CloudFormationTemplate dibuat menggunakan yang AWS CDK tersedia [di sini GitHub](#). Untuk informasi selengkapnya tentang AWS CDK, lihat [Panduan Developer AWS Cloud Development Kit \(AWS CDK\)](#).

3. Jalankan perintah berikut dari baris perintah, ganti *STACK_NAME* dengan nama unik untuk tumpukan.

Important

Nama tumpukan harus unik dalam AWS Wilayah dan AWS akun. Anda dapat menentukan hingga 128 karakter, dan angka serta tanda hubung diizinkan.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```


Untuk informasi selengkapnya tentang parameter `create-stack` perintah, lihat [panduan Referensi AWS CLI Perintah](#), dan [Panduan AWS CloudFormation Pengguna](#).

Untuk melihat sumber daya yang dibuat, buka AWS CloudFormation di konsol AWS manajemen, pilih tumpukan, dan pilih tab Sumber Daya.

Memahami aplikasi AWS Messaging

Untuk mengirim pesan ke antrian SQS, masukkan pesan ke dalam aplikasi dan pilih Kirim.

Setelah pesan dikirim, aplikasi menampilkan pesan.

Anda dapat memilih Purge untuk membersihkan pesan dari antrian Amazon SQS. Ini menghasilkan antrian kosong, dan tidak ada pesan yang ditampilkan dalam aplikasi.

Berikut ini menjelaskan bagaimana aplikasi menangani pesan:

- Pengguna memilih nama mereka dan memasukkan pesan mereka, dan mengirimkan pesan, yang memulai fungsi. `pushMessage`
- `pushMessage` mengambil Url Antrian Amazon SQS, lalu mengirim pesan dengan nilai ID pesan unik (GUID) teks pesan, dan pengguna ke Antrian Amazon SQS.
- `pushMessage` mengambil pesan dari Amazon SQS Queue, mengekstrak pengguna dan pesan untuk setiap pesan, dan menampilkan pesan.
- Pengguna dapat membersihkan pesan, yang menghapus pesan dari Amazon SQS Queue dan dari antarmuka pengguna.

Buat halaman HTML

Sekarang Anda membuat file HTML yang diperlukan untuk antarmuka pengguna grafis aplikasi (GUI). Buat file bernama `index.html`. Salin dan tempel kode di bawah ini ke `index.html`. Referensi HTML in `main.js`. Ini adalah versi bundel dari `index.js`, yang mencakup AWS SDK for JavaScript modul yang diperlukan.

```
<!doctype html>
<html
  xmlns:th="http://www.thymeleaf.org"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
>
```

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

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

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

    .messageelement img.right {
      float: right;
      margin-left: 20px;
      margin-right: 0;
    }
  </style>
```

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

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

    <div class="input-group">
      <div class="input-group-prepend">
        <span class="input-group-text">Message:</span>
      </div>
      <textarea
        class="form-control"
        id="textarea"
        aria-label="With textarea"
      ></textarea>
      <button
        type="button"
        onclick="pushMessage()"
        id="send"
        class="btn btn-success"
      >
        Send
      </button>
      <button
        type="button"
        onclick="purge()"
        id="refresh"
        class="btn btn-success"
      >
        Purge
      </button>
    </div>
    <!-- All of these child items are hidden and only displayed in a FancyBox
    ----->
```

```
<div id="hide" style="display: none">
  <div id="base" class="messageelement">
    
    <p id="text">Excellent! So, what do you want to do today?</p>
    <span class="time-right">11:02</span>
  </div>
</div>
</div>
</body>
</html>
```

Kode ini juga tersedia [di sini GitHub](#).

Membuat skrip browser

Dalam topik ini, Anda membuat skrip browser untuk aplikasi. Ketika Anda telah membuat skrip browser, Anda bundel ke dalam file yang disebut `main.js` seperti yang dijelaskan dalam [Bundling JavaScript](#).

Buat file bernama `index.js`. Salin dan tempel kode dari [sini GitHub](#) ke dalamnya.

Kode ini dijelaskan di bagian berikut:

1. [Konfigurasi](#)
2. [PopulateChat](#)
3. [pesan dorong](#)
4. [membersihkan](#)

Konfigurasi

Pertama, buat `libs` direktori buat objek klien Amazon SQS yang diperlukan dengan membuat file bernama `sqsClient.js` Ganti `REGION` dan `IDENTITY_POOL_ID` di masing-masing.

Note

Gunakan ID kumpulan identitas Amazon Cognito yang Anda buat. [Buat sumber AWS daya](#)

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

Di `index.js`, impor AWS SDK for JavaScript modul dan perintah yang diperlukan. Ganti `SQS_QUEUE_NAME` dengan *nama* Antrian Amazon SQS yang Anda buat di [Buat sumber AWS daya](#)

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

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

PopulateChat

`populateChatFungsi` onload secara otomatis mengambil URL untuk Antrian Amazon SQS, dan mengambil semua pesan dalam antrian, dan menampilkannya.

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

```
});

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

      // Loop through messages for user and message body.
      var i;
      for (i = 0; i < data.Messages.length; i++) {
        const name = data.Messages[i].MessageAttributes.Name.StringValue;
        const body = data.Messages[i].Body;
        // Create the HTML for the message.
        var userText = body + "<br><br><b>" + name;
        var myTextNode = $("#base").clone();
        myTextNode.text(userText);
        var image_url;
        var n = name.localeCompare("Scott");
        if (n == 0) image_url = "./images/av1.png";
        else image_url = "./images/av2.png";
        var images_div =
```

```

    '';
    myTextNode.html(userText);
    myTextNode.append(images_div);

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

```

Pesan push

Pengguna memilih nama mereka dan memasukkan pesan mereka, dan mengirimkan pesan, yang memulai fungsi. `pushMessage` mengambil URL Antrian Amazon SQS, lalu mengirim pesan dengan nilai ID pesan unik (GUID) teks pesan, dan pengguna ke Antrian Amazon SQS. Kemudian mengambil semua pesan dari Amazon SQS Queue dan menampilkannya.

```

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

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

  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,

```

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

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

// Retrieve messages from SQS Queue.
const final = await sqsClient.send(
  new ReceiveMessageCommand(getMessageParams)
);
console.log("Successfully retrieved", final.Messages);
$("#messages").empty();
// Loop through messages for user and message body.
var i;
for (i = 0; i < final.Messages.length; i++) {
  const name = final.Messages[i].MessageAttributes.Name.StringValue;
  const body = final.Messages[i].Body;
  // Create the HTML for the message.
```



```

    var userText = body + "<br><br><b>" + name;
    var myTextNode = $("#base").clone();
    myTextNode.text(userText);
    var image_url;
    var n = name.localeCompare("Scott");
    if (n == 0) image_url = "./images/av1.png";
    else image_url = "./images/av2.png";
    var images_div =
      '';
    myTextNode.html(userText);
    myTextNode.append(images_div);
    // Add the HTML to the GUI.
    $("#messages").append(myTextNode);
  }
} catch (err) {
  console.log("Error", err);
}
};
// Make the function available to the browser window.
window.pushMessage = pushMessage;

```

Bersihkan pesan

purgemenghapus pesan dari Amazon SQS Queue dan dari antarmuka pengguna.

```

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

```

```
);  
// Delete all the messages from the GUI.  
$("#messages").empty();  
console.log("Success. All messages deleted.", data);  
} catch (err) {  
  console.log("Error", err);  
}  
};  
  
// Make the function available to the browser window.  
window.purge = purge;
```

Bundling JavaScript

Kode skrip browser complete ini tersedia [di sini](#). GitHub .

Sekarang gunakan webpack untuk menggabungkan AWS SDK for JavaScript modul `index.js` dan ke dalam satu file, `main.js`.

1. Jika Anda belum melakukannya, ikuti contoh ini [Prasyarat](#) untuk menginstal webpack.

Note

Untuk informasi tentang webpack, lihat [Bundel aplikasi dengan webpack](#).

2. Jalankan yang berikut ini di baris perintah untuk menggabungkan JavaScript untuk contoh ini ke dalam file bernama `<index.js>`:

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

Langkah selanjutnya

Selamat! Anda telah membuat dan menerapkan aplikasi AWS Messaging yang menggunakan Amazon SQS. Seperti yang dinyatakan di awal tutorial ini, pastikan untuk menghentikan semua sumber daya yang Anda buat saat melalui tutorial ini untuk memastikan bahwa Anda tidak lagi dikenakan biaya untuk mereka. Anda dapat melakukan ini dengan menghapus AWS CloudFormation tumpukan yang Anda buat dalam [Buat sumber AWS daya](#) topik tutorial ini, sebagai berikut:

1. Buka [AWS CloudFormation di konsol AWS manajemen](#).
2. Buka halaman Stacks, dan pilih tumpukan.

3. Pilih Hapus.

Gunakan AWS Cloud9 dengan AWS SDK for JavaScript

Anda dapat menggunakan AWS Cloud9 dengan AWS SDK for JavaScript untuk menulis dan menjalankan kode browser Anda JavaScript — serta menulis, menjalankan, dan men-debug kode Node.js Anda — hanya menggunakan browser. AWS Cloud9 termasuk alat seperti editor kode dan terminal, ditambah debugger untuk kode Node.js.

Karena AWS Cloud9 IDE berbasis cloud, Anda dapat mengerjakan proyek Anda dari kantor, rumah, atau di mana saja menggunakan mesin yang terhubung ke internet. Untuk informasi umum tentang AWS Cloud9, lihat [Panduan AWS Cloud9 Pengguna](#).

Langkah-langkah berikut menjelaskan cara mengatur AWS Cloud9 dengan SDK untuk JavaScript.

Daftar Isi

- [Langkah 1: Siapkan AWS akun Anda untuk digunakan AWS Cloud9](#)
- [Langkah 2: Siapkan lingkungan AWS Cloud9 pengembangan Anda](#)
- [Langkah 3: Siapkan SDK untuk JavaScript](#)
 - [Untuk menyiapkan SDK JavaScript untuk Node.js](#)
 - [Untuk mengatur SDK untuk JavaScript di browser](#)
- [Langkah 4: Unduh kode contoh](#)
- [Langkah 5: Jalankan dan debug kode contoh](#)

Langkah 1: Siapkan AWS akun Anda untuk digunakan AWS Cloud9

Mulai gunakan AWS Cloud9 dengan masuk ke AWS Cloud9 konsol sebagai entitas AWS Identity and Access Management (IAM) (misalnya, pengguna IAM) yang memiliki izin akses AWS Cloud9 di akun Anda. AWS

Untuk menyiapkan entitas IAM di AWS akun Anda untuk diakses AWS Cloud9, dan untuk masuk ke AWS Cloud9 konsol, lihat [Penyiapan tim untuk AWS Cloud9](#) di Panduan AWS Cloud9 Pengguna.

Langkah 2: Siapkan lingkungan AWS Cloud9 pengembangan Anda

Setelah Anda masuk ke AWS Cloud9 konsol, gunakan konsol untuk membuat lingkungan AWS Cloud9 pengembangan. Setelah Anda membuat lingkungan, AWS Cloud9 buka IDE untuk lingkungan itu.

Lihat [Membuat lingkungan AWS Cloud9 di](#) Panduan AWS Cloud9 Pengguna untuk detailnya.

Note

Saat Anda membuat lingkungan di konsol untuk pertama kalinya, kami sarankan Anda memilih opsi untuk Membuat instance baru untuk lingkungan (EC2). Opsi ini memberi tahu AWS Cloud9 untuk membuat lingkungan, meluncurkan instans Amazon EC2, dan kemudian menghubungkan instance baru ke lingkungan baru. Ini adalah cara tercepat untuk mulai menggunakan AWS Cloud9.

Langkah 3: Siapkan SDK untuk JavaScript

Setelah AWS Cloud9 membuka IDE untuk lingkungan pengembangan Anda, ikuti salah satu atau kedua prosedur berikut untuk menggunakan IDE guna menyiapkan SDK JavaScript di lingkungan Anda.

Untuk menyiapkan SDK JavaScript untuk Node.js

1. Jika terminal belum terbuka di IDE, buka. Untuk melakukan ini, pada bilah menu di IDE, pilih Jendela, Terminal Baru.
2. Jalankan perintah berikut npm untuk digunakan untuk menginstal C1oud9 klien SDK untuk JavaScript.

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

Jika IDE tidak dapat menemukan npm, jalankan perintah berikut, satu per satu dalam urutan berikut, untuk menginstal npm. (Perintah ini mengasumsikan Anda memilih opsi untuk Membuat instance baru untuk lingkungan (EC2), sebelumnya dalam topik ini.)

⚠ 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](#) (Node Version Manager) GitHub .

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

Untuk mengatur SDK untuk JavaScript di browser

Untuk menggunakan SDK untuk JavaScript di halaman HTML Anda, gunakan WebPack untuk menggabungkan modul klien yang diperlukan dan semua JavaScript fungsi yang diperlukan ke dalam satu JavaScript file, dan tambahkan dalam tag skrip di <head> halaman HTML Anda. Sebagai contoh:

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

i Note

Untuk informasi selengkapnya tentang Webpack, lihat [Bundel aplikasi dengan webpack](#)

Langkah 4: Unduh kode contoh

Gunakan terminal yang Anda buka pada langkah sebelumnya untuk mengunduh kode contoh SDK JavaScript ke dalam lingkungan AWS Cloud9 pengembangan. (Jika terminal belum terbuka di IDE, buka dengan memilih Window, Terminal Baru pada bilah menu di IDE.)

Untuk mengunduh kode contoh, jalankan perintah berikut. Perintah ini mengunduh salinan semua contoh kode yang digunakan dalam dokumentasi AWS SDK resmi ke direktori root lingkungan Anda.

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

Untuk menemukan contoh kode untuk SDK for JavaScript, gunakan jendela Environment untuk membuka `ENVIRONMENT_NAME\aws-doc-sdk-examples\javascriptv3\example_code/src`, di mana `ENVIRONMENT_NAME` *adalah nama* lingkungan pengembangan Anda. AWS Cloud9

Untuk mempelajari cara bekerja dengan contoh kode ini dan lainnya, lihat [SDK untuk contoh JavaScript kode](#).

Langkah 5: Jalankan dan debug kode contoh

Untuk menjalankan kode di lingkungan AWS Cloud9 pengembangan Anda, lihat [Menjalankan kode Anda](#) di Panduan AWS Cloud9 Pengguna.

Untuk men-debug kode Node.js, lihat [Mendebug kode Anda](#) di AWS Cloud9 Panduan Pengguna.

SDK untuk contoh JavaScript kode (v3)

Contoh kode dalam topik ini menunjukkan cara menggunakan AWS SDK for JavaScript (v3) dengan AWS.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Contoh lintas layanan adalah contoh aplikasi yang bekerja di beberapa Layanan AWS.

Contoh-contoh

- [Tindakan dan skenario menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh lintas layanan menggunakan SDK untuk JavaScript \(v3\)](#)

Tindakan dan skenario menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Layanan AWS.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Layanan

- [Contoh Auto Scaling menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Amazon Bedrock menggunakan SDK for JavaScript \(v3\)](#)
- [Contoh Amazon Bedrock Runtime menggunakan SDK for JavaScript \(v3\)](#)
- [Agen untuk Amazon Bedrock contoh menggunakan SDK untuk JavaScript \(v3\)](#)

- [Agen untuk Amazon Bedrock Runtime contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [CloudWatch contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [CloudWatch Contoh acara menggunakan SDK for JavaScript \(v3\)](#)
- [CloudWatch Contoh log menggunakan SDK untuk JavaScript \(v3\)](#)
- [CodeBuild contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Penyedia Identitas Amazon Cognito menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh DynamoDB menggunakan SDK JavaScript for \(v3\)](#)
- [Contoh Amazon EC2 menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Elastic Load Balancing menggunakan SDK untuk JavaScript \(v3\)](#)
- [EventBridge contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [AWS Glue contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [HealthImaging contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh IAM menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Lambda menggunakan SDK untuk JavaScript \(v3\)](#)
- [Amazon Personalisasi contoh menggunakan SDK for JavaScript \(v3\)](#)
- [Contoh Personalisasi Acara Amazon menggunakan SDK for JavaScript \(v3\)](#)
- [Amazon Personalisasi contoh Runtime menggunakan SDK for \(v3\) JavaScript](#)
- [Amazon Pinpoint contoh menggunakan SDK for JavaScript \(v3\)](#)
- [Contoh Amazon Redshift menggunakan SDK for JavaScript \(v3\)](#)
- [Contoh Amazon S3 menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Gletser S3 menggunakan SDK untuk \(v3\) JavaScript](#)
- [SageMaker contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Secrets Manager menggunakan SDK for JavaScript \(v3\)](#)
- [Amazon SES contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Amazon SNS menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Amazon SQS menggunakan SDK untuk JavaScript \(v3\)](#)
- [Contoh Step Functions menggunakan SDK for JavaScript \(v3\)](#)
- [AWS STS contoh menggunakan SDK untuk JavaScript \(v3\)](#)
- [AWS Support contoh menggunakan SDK untuk JavaScript \(v3\)](#)

- [Contoh Amazon Transcribe menggunakan SDK for JavaScript \(v3\)](#)

Contoh Auto Scaling menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Auto Scaling.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Lampirkan grup target ELB ke grup Auto Scaling

Contoh kode berikut menunjukkan cara melampirkan grup target ELB ke grup Auto Scaling.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const client = new AutoScalingClient({});  
await client.send(  

```

```
new AttachLoadBalancerTargetGroupsCommand({
  AutoScalingGroupName: NAMES.autoScalingGroupName,
  TargetGroupARNs: [state.targetGroupArn],
}),
);
```

- Untuk detail API, lihat [AttachLoadBalancerTargetGroups](#) di Referensi AWS SDK for JavaScript API.

Skenario

Membangun dan mengelola layanan yang tangguh

Contoh kode berikut menunjukkan cara membuat layanan web load-balanced yang mengembalikan rekomendasi buku, film, dan lagu. Contoh ini menunjukkan cara layanan tersebut merespons kegagalan, serta cara merestrukturisasi layanan agar lebih tangguh ketika terjadi kegagalan.

- Menggunakan grup Amazon EC2 Auto Scaling untuk membuat instans Amazon Elastic Compute Cloud (Amazon EC2) berdasarkan templat peluncuran dan menyimpan sejumlah instans dalam rentang yang ditentukan.
- Menangani dan mendistribusikan permintaan HTTP dengan Elastic Load Balancing.
- Memantau kondisi instans dalam grup Auto Scaling dan meneruskan permintaan hanya ke instans yang sehat.
- Menjalankan server web Python pada setiap instans EC2 untuk menangani permintaan HTTP. Server web merespons dengan memberikan rekomendasi dan melakukan pemeriksaan kondisi.
- Menyimulasikan layanan yang direkomendasikan dengan tabel Amazon DynamoDB.
- Kontrol respons server web terhadap permintaan dan pemeriksaan kesehatan dengan memperbarui AWS Systems Manager parameter.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menjalankan skenario interaktif di prompt perintah.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 * - deploy
 * - demo
 * - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};
```

```
// Call function if run directly
import { fileURLToPath } from "url";

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

Menyusun langkah-langkah untuk men-deploy semua sumber daya.

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

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
```

```
    CreateAutoScalingGroupCommand,
    AutoScalingClient,
    AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
    CreateListenerCommand,
    CreateLoadBalancerCommand,
    CreateTargetGroupCommand,
    ElasticLoadBalancingV2Client,
    waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
    ScenarioOutput,
    ScenarioInput,
    ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
    new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
    new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
        type: "confirm",
    }),
    new ScenarioAction(
        "handleConfirmDeployment",
        (c) => c.confirmDeployment === false && process.exit(),
    ),
    new ScenarioOutput(
        "creatingTable",
        MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
    new ScenarioAction("createTable", async () => {
        const client = new DynamoDBClient({});
        await client.send(
            new CreateTableCommand({
                TableName: NAMES.tableName,
                ProvisionedThroughput: {
```

```
        ReadCapacityUnits: 5,
        WriteCapacityUnits: 5,
    },
    AttributeDefinitions: [
        {
            AttributeName: "MediaType",
            AttributeType: "S",
        },
        {
            AttributeName: "ItemId",
            AttributeType: "N",
        },
    ],
    KeySchema: [
        {
            AttributeName: "MediaType",
            KeyType: "HASH",
        },
        {
            AttributeName: "ItemId",
            KeyType: "RANGE",
        },
    ],
    }),
);
await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
        readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );
});
```

```
return client.send(
  new BatchWriteItemCommand({
    RequestItems: {
      [NAMES.tableName]: recommendations.map((item) => ({
        PutRequest: { Item: item },
      })),
    },
  }),
);
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
  const { KeyMaterial } = await client.send(
    new CreateKeyPairCommand({
      KeyName: NAMES.keyPairName,
    }),
  );
  writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
```



```
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  )),
);
state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  ),
});
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
```

```
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
```

```

new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
new ScenarioAction("createLaunchTemplate", async () => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(
    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
      LaunchTemplateData: {
        InstanceType: "t3.micro",
        ImageId: Parameter.Value,
        IamInstanceProfile: { Name: NAMES.instanceProfileName },
        UserData: readFileSync(
          join(RESOURCES_PATH, "server_startup_script.sh"),
        ).toString("base64"),
        KeyName: NAMES.keyPairName,
      },
    }),
  );
}

```

```
    }},
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({
        AvailabilityZones: state.availabilityZoneNames,
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
          LaunchTemplateName: NAMES.launchTemplateName,
          Version: "$Default",
        },
        MinSize: 3,
        MaxSize: 3,
      }),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
```

```

    */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
    ),
    new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
      type: "confirm",
    }),
    new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
    new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
    new ScenarioAction("getVpc", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
      const client = new EC2Client({});
      const { Vpcs } = await client.send(
        new DescribeVpcsCommand({
          Filters: [{ Name: "is-default", Values: ["true"] }],
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
      state.defaultVpc = Vpcs[0].VpcId;
    }),
    new ScenarioOutput("gotVpc", (state) =>
      MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
    ),
    new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
    new ScenarioAction("getSubnets", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
      const client = new EC2Client({});
      const { Subnets } = await client.send(
        new DescribeSubnetsCommand({
          Filters: [
            { Name: "vpc-id", Values: [state.defaultVpc] },
            { Name: "availability-zone", Values: state.availabilityZoneNames },
            { Name: "default-for-az", Values: ["true"] },
          ],
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
      state.subnets = Subnets.map((subnet) => subnet.SubnetId);
    }),
  },

```

```
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
  state.targetGroupProtocol = targetGroup.Protocol;
  state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
  "createdLoadBalancerTargetGroup",
  MESSAGES.createdLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
```

```
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
    }),
  );
  state.loadBalancerDns = LoadBalancers[0].DNSName;
  state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
  await waitUntilLoadBalancerAvailable(
    { client },
    { Names: [NAMES.loadBalancerName] },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
  MESSAGES.createdLoadBalancer
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioOutput(
  "creatingListener",
  MESSAGES.creatingLoadBalancerListener
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
),
new ScenarioAction("createListener", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
  const client = new ElasticLoadBalancingV2Client({});
  const { Listeners } = await client.send(
    new CreateListenerCommand({
      LoadBalancerArn: state.loadBalancerArn,
      Protocol: state.targetGroupProtocol,
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  );
}),
```

```

    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }},
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }},
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
    state
    */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({

```



```

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

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

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

```

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

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

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

Menyusun langkah-langkah untuk menjalankan demo.

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

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
```

```
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    }
  }
);
```

```
    }
  } else {
    throw new Error(MESSAGES.demoFindLoadBalancerError);
  }
},
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
  balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
  }
);
```

```
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];
```

```
/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
```

```

    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,
        Value: "static",
        Overwrite: true,
        Type: "String",
      }),
    );
  }
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
  "badCredentialsConfirmation",
  MESSAGES.demoBadCredentialsConfirmation,
  { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
  if (!state.badCredentialsConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("fixDynamoDBName", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioAction(
  "badCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
   */
  async (state) => {
    await createSsmOnlyInstanceProfile();
    const autoScalingClient = new AutoScalingClient({});
    const { AutoScalingGroups } = await autoScalingClient.send(

```



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

await ec2Client.send(
  new RebootInstancesCommand({
    InstanceIds: [state.targetInstance.InstanceId],
  }),
);

const ssmClient = new SSMClient({});
await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
  const { InstanceInformationList } = await ssmClient.send(
    new DescribeInstanceInformationCommand({}),
  );
```

```

    const instance = InstanceInformationList.find(
      (info) => info.InstanceId === state.targetInstance.InstanceId,
    );

    if (!instance) {
      throw new Error("Instance not found.");
    }
  });

  await ssmClient.send(
    new SendCommandCommand({
      InstanceIds: [state.targetInstance.InstanceId],
      DocumentName: "AWS-RunShellScript",
      Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
    }),
  );
},
),
new ScenarioOutput(
  "testBadCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
   */
  (state) =>
    MESSAGES.demoTestBadCredentials.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
),
loadBalancerLoop,
new ScenarioInput(
  "deepHealthCheckConfirmation",
  MESSAGES.demoDeepHealthCheckConfirmation,
  { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
  if (!state.deepHealthCheckConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("deepHealthCheck", async () => {
  const client = new SSMClient({});
  await client.send(

```

```

    new PutParameterCommand({
      Name: NAMES.ssmHealthCheckKey,
      Value: "deep",
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "killInstanceConfirmation",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
  ssm').InstanceInformation }} state
   */
  (state) =>
    MESSAGES.demoKillInstanceConfirmation.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
  { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
  if (!state.killInstanceConfirmation) {
    process.exit();
  }
}),
new ScenarioAction(
  "killInstance",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
  ssm').InstanceInformation }} state
   */
  async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new TerminateInstanceInAutoScalingGroupCommand({
        InstanceId: state.targetInstance.InstanceId,
        ShouldDecrementDesiredCapacity: false,
      }),
    );
  },
),
},

```

```
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("failOpen", () => {
  const client = new SSMClient({});
  return client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: `fake-table-${Date.now()}`,
      Overwrite: true,
      Type: "String",
    })
  );
}),
new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "resetTableConfirmation",
  MESSAGES.demoResetTableConfirmation,
  { type: "confirm" },
),
new ScenarioAction("resetTableExit", (state) => {
  if (!state.resetTableConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("resetTable", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
```

```
    }),
  );
}),
new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
healthCheckLoop,
loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    ));
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
}
```

```

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

return InstanceProfile;
}

```

Menyusun langkah-langkah untuk menghancurkan semua sumber daya.

```

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

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,

```

```
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
```

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

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



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

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

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

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

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

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

```
// snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
const client = new ElasticLoadBalancingV2Client({});
try {
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    client.send(
      new DeleteTargetGroupCommand({
        TargetGroupArn: TargetGroups[0].TargetGroupArn,
      }),
    ),
  );
} catch (e) {
  state.deleteLoadBalancerTargetGroupError = e;
}
// snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  } else {
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  }
});
```

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

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



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

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

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

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

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

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

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

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for JavaScript .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)
 - [DescribeInstanceProfileAssociations](#)
 - [DescribeInstances](#)
 - [DescribeLoadBalancers](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGroups](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)
 - [ReplaceInstanceProfileAssociation](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

Contoh Amazon Bedrock menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon Bedrock.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Amazon Bedrock

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon Bedrock.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

import { fileURLToPath } from "url";

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

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

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

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

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

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

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

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

    ${REGION}.`,
  );

  return response;
};

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

- Untuk detail API, lihat [ListFoundationModels](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)

Tindakan

Dapatkan detail tentang model fondasi Amazon Bedrock

Contoh kode berikut menunjukkan cara mendapatkan detail tentang model pondasi Amazon Bedrock.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan detail tentang model pondasi.

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

import { fileURLToPath } from "url";

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

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 * models.
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();

  const command = new GetFoundationModelCommand({
```

```
    modelIdentifier: "amazon.titan-embed-text-v1",
  });

  const response = await client.send(command);

  return response.modelDetails;
};


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

- Untuk detail API, lihat [GetFoundationModel](#) di Referensi AWS SDK for JavaScript API.

Daftar model yayasannya Amazon Bedrock yang tersedia

Contoh kode berikut menunjukkan cara membuat daftar model dasar Amazon Bedrock yang tersedia.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar model pondasi yang tersedia.

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

import { fileURLToPath } from "url";

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

/**
```



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

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

  const command = new ListFoundationModelsCommand(input);

  const response = await client.send(command);

  return response.modelSummaries;
};

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

- Untuk detail API, lihat [ListFoundationModels](#) di Referensi AWS SDK for JavaScript API.

Contoh Amazon Bedrock Runtime menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon Bedrock Runtime.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Pembuatan teks dengan AI21 Labs Jurassic-2

Contoh kode berikut menunjukkan cara menjalankan model AI21 Labs Jurassic-2 di Amazon Bedrock untuk pembuatan teks.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan model dasar AI21 Labs Jurassic-2 untuk menghasilkan teks.

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

import { fileURLToPath } from "url";

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

/**
 * @typedef {Object} Data
 * @property {string} text
 *
 * @typedef {Object} Completion
 * @property {Data} data
 */
```

```
* @typedef {Object} ResponseBody
* @property {Completion[]} completions
*/

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

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

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

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

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

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

    return responseBody.completions[0].data.text;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
```

```
    console.error(
      `Access denied. Ensure you have the correct permissions to invoke
      ${modelId}.`,
    );
  } else {
    throw err;
  }
}
};

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

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

- Untuk detail API, lihat [InvokeModel](#) di Referensi AWS SDK for JavaScript API.

Pembuatan teks dengan Amazon Titan Text G1

Contoh kode berikut menunjukkan cara memanggil model Amazon Titan Text G1 di Amazon Bedrock untuk pembuatan teks.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Panggil model dasar Amazon Titan Text G1 untuk menghasilkan teks.

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

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

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

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

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

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

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

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

  const command = new InvokeModelCommand({
```

```
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

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

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

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

```
Miguel can you document any other concerns so that I can discuss with James only once?
```

```
Miguel: Sure.
```

```
From the meeting transcript above, Create a list of action items for each person.`;
```

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

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

- Untuk detail API, lihat [InvokeModel](#) di Referensi AWS SDK for JavaScript API.

Pembuatan teks dengan Anthropic Claude 2

Contoh kode berikut menunjukkan cara memanggil model Anthropic Claude 2 di Amazon Bedrock untuk pembuatan teks.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan model dasar Anthropic Claude 2 untuk menghasilkan teks.

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

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
```

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

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

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

  const modelId = "anthropic.claude-v2";

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

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Anthropic Claude, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-claude.html
   */
  const payload = {
    prompt: enclosedPrompt,
    max_tokens_to_sample: 500,
    temperature: 0.5,
    stop_sequences: ["\n\nHuman:"],
  };

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

  try {
```



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

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

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

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

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

- Untuk detail API, lihat [InvokeModelId](#) Referensi AWS SDK for JavaScript API.

Pembuatan teks dengan Meta Llama 2 Chat

Contoh kode berikut menunjukkan cara memanggil model Obrolan Meta Llama 2 di Amazon Bedrock untuk pembuatan teks.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Panggil model dasar Meta Llama 2 Chat untuk menghasilkan teks.

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

import { fileURLToPath } from "url";

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

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

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

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

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

```
    prompt,
    temperature: 0.5,
    top_p: 0.9,
    max_gen_len: 512,
  };

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

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

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

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

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

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

- Untuk detail API, lihat [InvokeModel](#) di Referensi AWS SDK for JavaScript API.

Pembuatan teks dengan Mistral 7B

Contoh kode berikut menunjukkan cara memanggil model model Mistral 7B di Amazon Bedrock untuk pembuatan teks.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Panggil model pondasi Mistral 7B untuk menghasilkan teks.

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

import { fileURLToPath } from "url";

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

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

/**
 * Invokes the Mistral 7B model to run an inference using the input
 * provided in the request body.
 */
```

```
* @param {string} prompt - The prompt that you want Mistral to complete.
* @returns {string[]} A list of inference responses (completions) from the model.
*/
export const invokeMistral7B = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-west-2" });

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

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

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

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

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

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

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

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

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

- Untuk detail API, lihat [InvokeModel](#) di Referensi AWS SDK for JavaScript API.

Pembuatan teks dengan Mixtral 8x7B

Contoh kode berikut menunjukkan cara memanggil model model Mixtral 8x7B di Amazon Bedrock untuk pembuatan teks.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Panggil model dasar Mixtral 8x7B untuk menghasilkan teks.

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

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
```

```
} from "@aws-sdk/client-bedrock-runtime";
import { invokeMistral7B } from "./invoke-mistral7b.js";

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

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

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `
```

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

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

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

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

- Untuk detail API, lihat [InvokeModel](#) di Referensi AWS SDK for JavaScript API.

Agen untuk Amazon Bedrock contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Agen untuk Amazon Bedrock.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Agen untuk Amazon Bedrock

Contoh kode berikut menunjukkan cara memulai menggunakan Agen untuk Amazon Bedrock.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

import { fileURLToPath } from "url";

import {
  BedrockAgentClient,
  GetAgentCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * @typedef {Object} AgentSummary
 */

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 * Agents Client.
```

```
*
* This function first initializes the Amazon Bedrock Agents client for a specific
region.
* It then retrieves a list of existing agents using the streamlined paginator
approach.
* For each agent found, it retrieves detailed information using a command object.
*
* Demonstrates:
* - Use of the Bedrock Agents client to initialize and communicate with the AWS
service.
* - Listing resources in a paginated response pattern.
* - Accessing an individual resource using a command object.
*
* @returns {Promise<void>} A promise that resolves when the function has completed
execution.
*/
export const main = async () => {
  const region = "us-east-1";

  console.log("=".repeat(68));

  console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
  const client = new BedrockAgentClient({ region });

  console.log(`Retrieving the list of existing agents...`);
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});

  /** @type {AgentSummary[]} */
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  console.log(`Found ${agentSummaries.length} agents in ${region}.`);

  if (agentSummaries.length > 0) {
    for (const agentSummary of agentSummaries) {
      const agentId = agentSummary.agentId;
      console.log("=".repeat(68));
      console.log(`Retrieving agent with ID: ${agentId}:`);
      console.log("-".repeat(68));

      const command = new GetAgentCommand({ agentId });
```

```
const response = await client.send(command);
const agent = response.agent;

console.log(` Name: ${agent.agentName}`);
console.log(` Status: ${agent.agentStatus}`);
console.log(` ARN: ${agent.agentArn}`);
console.log(` Foundation model: ${agent.foundationModel}`);
}
}
console.log("=".repeat(68));
};

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

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [GetAgent](#)
 - [ListAgents](#)

Topik

- [Tindakan](#)

Tindakan

Buat agen

Contoh kode berikut menunjukkan cara membuat agen Amazon Bedrock.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat agen.

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  CreateAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Creates an Amazon Bedrock Agent.
 *
 * @param {string} agentName - A name for the agent that you create.
 * @param {string} foundationModel - The foundation model to be used by the agent
you create.
 * @param {string} agentResourceRoleArn - The ARN of the IAM role with permissions
required by the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
containing details of the created agent.
 */
export const createAgent = async (
  agentName,
  foundationModel,
  agentResourceRoleArn,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  const command = new CreateAgentCommand({
    agentName,
    foundationModel,
    agentResourceRoleArn,
  });
  const response = await client.send(command);

  return response.agent;
};

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

```
// Replace the placeholders for agentName and accountId, and roleName with a
unique name for the new agent,
// the id of your AWS account, and the name of an existing execution role that the
agent can use inside your account.
// For foundationModel, specify the desired model. Ensure to remove the brackets
'[]' before adding your data.

// A string (max 100 chars) that can include letters, numbers, dashes '-', and
underscores '_'.
const agentName = "[your-bedrock-agent-name]";

// Your AWS account id.
const accountId = "[123456789012]";

// The name of the agent's execution role. It must be prefixed by
`AmazonBedrockExecutionRoleForAgents_`.
const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

// The ARN for the agent's execution role.
// Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

// Specify the model for the agent. Change if a different model is preferred.
const foundationModel = "anthropic.claude-v2";

// Check for unresolved placeholders in agentName and roleArn.
checkForPlaceholders([agentName, roleArn]);

console.log(`Creating a new agent...`);

const agent = await createAgent(agentName, foundationModel, roleArn);
console.log(agent);
}
```

- Untuk detail API, lihat [CreateAgent](#) di Referensi AWS SDK for JavaScript API.

Hapus agen

Contoh kode berikut menunjukkan cara menghapus agen Amazon Bedrock.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus agen.

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent to delete.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
  and some additional metadata.
 */
export const deleteAgent = (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });
  const command = new DeleteAgentCommand({ agentId });
  return client.send(command);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets (`[]`) before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";
}
```

```
// Check for unresolved placeholders in agentId.
checkForPlaceholders([agentId]);

console.log(`Deleting agent with ID ${agentId}...`);

const response = await deleteAgent(agentId);
console.log(response);
}
```

- Untuk detail API, lihat [DeleteAgent](#) di Referensi AWS SDK for JavaScript API.

Dapatkan informasi tentang agen

Contoh kode berikut menunjukkan cara mendapatkan informasi tentang agen Amazon Bedrock.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan agen.

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 */
```

```
* @param {string} [region='us-east-1'] - The AWS region in use.
* @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
containing the agent details.
*/
export const getAgent = async (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const command = new GetAgentCommand({ agentId });
  const response = await client.send(command);
  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Retrieving agent with ID ${agentId}...`);

  const agent = await getAgent(agentId);
  console.log(agent);
}
```

- Untuk detail API, lihat [GetAgent](#) di Referensi AWS SDK for JavaScript API.

Buat daftar grup aksi untuk agen

Contoh kode berikut menunjukkan cara membuat daftar grup tindakan untuk agen Amazon Bedrock.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar grup aksi untuk agen.

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  ListAgentActionGroupsCommand,
  paginateListAgentActionGroups,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 * pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  // Create a paginator configuration
  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const params = { agentId, agentVersion };

  const pages = paginateListAgentActionGroups(paginatorConfig, params);

  // Paginate until there are no more results
```

```
const actionGroupSummaries = [];
for await (const page of pages) {
  actionGroupSummaries.push(...page.actionGroupSummaries);
}

return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 * ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 * and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 * away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const actionGroupSummaries = [];
  do {
    const command = new ListAgentActionGroupsCommand({
      agentId,
      agentVersion,
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
    const response = await client.send(command);

    for (const actionGroup of response.actionGroupSummaries || []) {
```

```
    actionGroupSummaries.push(actionGroup);
  }

  nextToken = response.nextToken;
} while (nextToken);

return actionGroupSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId and agentVersion with an existing agent's
  id and version.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
  'DRAFT').
  const agentVersion = "[DRAFT]";

  // Check for unresolved placeholders in agentId and agentVersion.
  checkForPlaceholders([agentId, agentVersion]);

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using ListAgentActionGroupsCommand:",
  );

  for (const actionGroup of await listAgentActionGroupsWithCommandObject(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using the paginateListAgents function:",
  );
  for (const actionGroup of await listAgentActionGroupsWithPaginator(
    agentId,
    agentVersion,
```

```
    )) {  
      console.log(actionGroup);  
    }  
  }  
}
```

- Untuk detail API, lihat [ListAgentActionGroups](#) di Referensi AWS SDK for JavaScript API.

Daftar agen yang tersedia

Contoh kode berikut menunjukkan cara membuat daftar Agen untuk Amazon Bedrock milik akun.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar agen milik akun.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
  
import { fileURLToPath } from "url";  
  
import {  
  BedrockAgentClient,  
  ListAgentsCommand,  
  paginateListAgents,  
} from "@aws-sdk/client-bedrock-agent";  
  
/**  
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator  
 * function.  
 *  
 * This function leverages a paginator, which abstracts the complexity of  
 * pagination, providing  
 * a straightforward way to handle paginated results inside a `for await...of` loop.  
 *  
 * @param {string} [region='us-east-1'] - The AWS region in use.
```

```
* @returns {Promise<AgentSummary[]>} An array of agent summaries.
*/
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 * ListAgentsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 * and processing the response.
 *
 * Pagination must manually be managed. For a simplified approach that abstracts
 * away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const agentSummaries = [];
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });
  };
```

```
/** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
const paginatedResponse = await client.send(command);

agentSummaries.push(...(paginatedResponse.agentSummaries || []));

nextToken = paginatedResponse.nextToken;
} while (nextToken);

return agentSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
  console.log("Listing agents using ListAgentsCommand:");
  for (const agent of await listAgentsWithCommandObject()) {
    console.log(agent);
  }

  console.log("=".repeat(68));
  console.log("Listing agents using the paginateListAgents function:");
  for (const agent of await listAgentsWithPaginator()) {
    console.log(agent);
  }
}
```

- Untuk detail API, lihat [ListAgents](#) di Referensi AWS SDK for JavaScript API.

Agen untuk Amazon Bedrock Runtime contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Agents for Amazon Bedrock Runtime.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Meminta agen

Contoh kode berikut menunjukkan cara memanggil agen Amazon Bedrock.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

import {
  BedrockAgentRuntimeClient,
  InvokeAgentCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

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

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
```

```
* @param {string} sessionId - An arbitrary identifier for the session.
*/
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
  // });

  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (let chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
      console.log(chunk);
      const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
      completion += decodedResponse;
    }

    return { sessionId: sessionId, completion };
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly
```



```
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- Untuk detail API, lihat [InvokeAgent](#) di Referensi AWS SDK for JavaScript API.

CloudWatch contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) with CloudWatch.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik


- [Tindakan](#)

Tindakan

Buat alarm metrik

Contoh kode berikut menunjukkan cara membuat atau memperbarui CloudWatch alarm Amazon dan mengaitkannya dengan metrik yang ditentukan, ekspresi matematika metrik, model deteksi anomali, atau kueri Wawasan Metrik.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
        the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};
```


```
export default run();
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";  
  
export const client = new CloudWatchClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutMetricAlarm](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create CloudWatch service object  
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });  
  
var params = {  
  AlarmName: "Web_Server_CPU_Utilization",  
  ComparisonOperator: "GreaterThanThreshold",  
  EvaluationPeriods: 1,  
  MetricName: "CPUUtilization",  
  Namespace: "AWS/EC2",  
  Period: 60,  
  Statistic: "Average",  
  Threshold: 70.0,  
  ActionsEnabled: false,  
  AlarmDescription: "Alarm when server CPU exceeds 70%",  
  Dimensions: [  

```

```
{
  Name: "InstanceId",
  Value: "INSTANCE_ID",
},
],
Unit: "Percent",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutMetricAlarm](#) di Referensi AWS SDK for JavaScript API.

Menghapus alarm

Contoh kode berikut menunjukkan cara menghapus CloudWatch alarm Amazon.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });
```

```
try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteAlarms](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

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

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

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteAlarms](#) di Referensi AWS SDK for JavaScript API.

Jelaskan alarm untuk metrik

Contoh kode berikut menunjukkan cara mendeskripsikan CloudWatch alarm Amazon untuk metrik.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};
```


```
export default run();
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";  
  
export const client = new CloudWatchClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DescribeAlarmsForMetric](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create CloudWatch service object  
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });  
  
cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {  
  if (err) {  
    console.log("Error", err);  
  } else {  
    // List the names of all current alarms in the console  
    data.MetricAlarms.forEach(function (item, index, array) {  
      console.log(item.AlarmName);  
    });  
  }  
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DescribeAlarmsForMetric](#) di Referensi AWS SDK for JavaScript API.

Menonaktifkan tindakan alarm

Contoh kode berikut menunjukkan cara menonaktifkan tindakan CloudWatch alarm Amazon.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```


- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DisableAlarmActions](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

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

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

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DisableAlarmActions](#) di Referensi AWS SDK for JavaScript API.

Mengaktifkan tindakan alarm

Contoh kode berikut menunjukkan cara mengaktifkan tindakan CloudWatch alarm Amazon.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [EnableAlarmActions](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

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

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

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

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

```
console.log("Alarm action added", data);
var paramsEnableAlarmAction = {
  AlarmNames: [params.AlarmName],
};
cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Alarm action enabled", data);
  }
});
}
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [EnableAlarmActions](#) di Referensi AWS SDK for JavaScript API.

Membuat daftar metrik

Contoh kode berikut menunjukkan cara membuat daftar metadata untuk metrik Amazon CloudWatch. Untuk mendapatkan data untuk metrik, gunakan `GetMetricStatistics` tindakan `GetMetricData` atau.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = () => {
  // Use the AWS console to see available namespaces and metric names. Custom
  metrics can also be created.
```

```
// https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
const command = new ListMetricsCommand({
  Dimensions: [
    {
      Name: "LogGroupName",
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
});

return client.send(command);
};
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListMetrics](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListMetrics](#) di Referensi AWS SDK for JavaScript API.

Masukkan data ke dalam metrik

Contoh kode berikut menunjukkan cara mempublikasikan titik data metrik ke Amazon CloudWatch.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
```

```
// See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
// and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
// for more information about the parameters in this command.
const command = new PutMetricDataCommand({
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
});

try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```


Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutMetricData](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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


- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutMetricData](#) di Referensi AWS SDK for JavaScript API.

CloudWatch Contoh acara menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan CloudWatch Events.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Menambahkan target

Contoh kode berikut menunjukkan cara menambahkan target ke CloudWatch acara Amazon Events.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";
```

```
const run = async () => {
  const command = new PutTargetsCommand({
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,

    // The targets to add to the rule.
    Targets: [
      {
        Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
        // The ID of the target. Choose a unique ID for each target.
        Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutTargets](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutTargets](#) di Referensi AWS SDK for JavaScript API.

Buat aturan terjadwal

Contoh kode berikut menunjukkan cara membuat aturan terjadwal CloudWatch Acara Amazon.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  // Request parameters for PutRule.
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/API_PutRule.html#API_PutRule_RequestParameters
  const command = new PutRuleCommand({
    Name: process.env.CLOUDWATCH_EVENTS_RULE,

    // The event pattern for the rule.
    // Example: {"source": ["my.app"]}
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,

    // The state of the rule. Valid values: ENABLED, DISABLED
    State: "ENABLED",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutRule](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutRule](#) di Referensi AWS SDK for JavaScript API.

Kirim acara

Contoh kode berikut menunjukkan cara mengirim CloudWatch acara Amazon Events.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
        Source: "my.app",

        // The name of the event that is being sent.
        DetailType: "My Custom Event",

        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutEvents](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutEvents](#) di Referensi AWS SDK for JavaScript API.

CloudWatch Contoh log menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan CloudWatch Log.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Membuat grup log

Contoh kode berikut menunjukkan cara membuat grup CloudWatch log Log baru.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh selengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
```



```
import { client } from "../libs/client.js";

const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Untuk detail API, lihat [CreateLogGroup](#) di Referensi AWS SDK for JavaScript API.

Buat filter langganan

Contoh kode berikut menunjukkan cara membuat filter langganan Amazon CloudWatch Logs.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
    // An ARN of a same-account Kinesis stream, Kinesis Firehose
    // delivery stream, or Lambda function.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
    SubscriptionFilters.html
    destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,
```

```

// A name for the filter.
filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

// A filter pattern for subscribing to a filtered stream of log events.
// https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
FilterAndPatternSyntax.html
filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

// The name of the log group. Messages in this group matching the filter pattern
// will be sent to the destination ARN.
logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
});

try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();

```

- Untuk detail API, lihat [PutSubscriptionFilter](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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

// Create the CloudWatchLogs service object
var cw1 = 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 informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutSubscriptionFilter](#) di Referensi AWS SDK for JavaScript API.

Menghapus grup log

Contoh kode berikut menunjukkan cara menghapus grup CloudWatch log Log yang ada.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  }
};
```

```
    } catch (err) {  
      console.error(err);  
    }  
  };  
  
export default run();
```

- Untuk detail API, lihat [DeleteLogGroup](#) di Referensi AWS SDK for JavaScript API.

Hapus filter langganan

Contoh kode berikut menunjukkan cara menghapus filter langganan Amazon CloudWatch Logs.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";  
import { client } from "../libs/client.js";  
  
const run = async () => {  
  const command = new DeleteSubscriptionFilterCommand({  
    // The name of the filter.  
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,  
    // The name of the log group.  
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,  
  });  
  
  try {  
    return await client.send(command);  
  } catch (err) {  
    console.error(err);  
  }  
};  
  
export default run();
```

- Untuk detail API, lihat [DeleteSubscriptionFilter](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the CloudWatchLogs service object
var cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

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


cw1.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteSubscriptionFilter](#) di Referensi AWS SDK for JavaScript API.

Jelaskan filter langganan yang ada

Contoh kode berikut menunjukkan cara mendeskripsikan filter langganan Amazon CloudWatch Logs yang ada.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";


const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Untuk detail API, lihat [DescribeSubscriptionFilters](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DescribeSubscriptionFilters](#) di Referensi AWS SDK for JavaScript API.

Jelaskan grup log

Contoh kode berikut menunjukkan bagaimana mendeskripsikan grup CloudWatch log Log.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";
```

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

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups && page.logGroups.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }

  console.log(logGroups);
  return logGroups;
};
```

- Untuk detail API, lihat [DescribeLogGroups](#) di Referensi AWS SDK for JavaScript API.

Dapatkan hasil kueri

Contoh kode berikut menunjukkan bagaimana untuk mendapatkan hasil query.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
```

- Untuk detail API, lihat [GetQueryResults](#) di Referensi AWS SDK for JavaScript API.

Mulai sesi Live Tail

Contoh kode berikut menunjukkan cara memulai sesi Live Tail untuk grup log/aliran log yang ada.

SDK untuk JavaScript (v3)

Sertakan file-file yang diperlukan.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-cloudwatch-logs";
```

Tangani acara dari sesi Live Tail.

```
async function handleResponseAsync(response) {
  try {
    for await (const event of response.responseStream) {
      if (event.sessionStart !== undefined) {
        console.log(event.sessionStart);
      } else if (event.sessionUpdate !== undefined) {
        for (const logEvent of event.sessionUpdate.sessionResults) {
          const timestamp = logEvent.timestamp;
          const date = new Date(timestamp);
          console.log "[" + date + "]" + logEvent.message);
        }
      } else {
        console.error("Unknown event type");
      }
    }
  } catch (err) {
    // On-stream exceptions are captured here
    console.error(err)
  }
}
```

Mulai sesi Live Tail.

```
const client = new CloudWatchLogsClient();

const command = new StartLiveTailCommand({
  logGroupIdentifiers: logGroupIdentifiers,
```

```
    logStreamNames: logStreamNames,  
    logEventFilterPattern: filterPattern  
  });  
  try{  
    const response = await client.send(command);  
    handleResponseAsync(response);  
  } catch (err){  
    // Pre-stream exceptions are captured here  
    console.log(err);  
  }
```

Hentikan sesi Live Tail setelah periode waktu berlalu.

```
/* Set a timeout to close the client. This will stop the Live Tail session. */  
setTimeout(function() {  
  console.log("Client timeout");  
  client.destroy();  
}, 10000);
```

- Untuk detail API, lihat [StartLiveTail](#) di Referensi AWS SDK for JavaScript API.

Mulai kueri

Contoh kode berikut menunjukkan bagaimana memulai query.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/**  
 * Wrapper for the StartQueryCommand. Uses a static query string  
 * for consistency.  
 * @param {[Date, Date]} dateRange  
 * @param {number} maxLogs  
 * @returns {Promise<{ queryId: string }>}
```

```
*/
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
      // before the log group was created.
      throw new DateOutOfBoundsError(message);
    }

    throw err;
  }
}
```

- Untuk detail API, lihat [StartQuery](#) di Referensi AWS SDK for JavaScript API.

Skenario

Jalankan kueri besar

Contoh kode berikut menunjukkan cara menggunakan CloudWatch Log untuk menanyakan lebih dari 10.000 catatan.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Ini adalah titik masuknya.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "../cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}

const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
  logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
  dateRange: [
    new Date(parseInt(process.env.QUERY_START_DATE)),
    new Date(parseInt(process.env.QUERY_END_DATE)),
  ],
});

await cloudWatchQuery.run();

console.log(
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found:
  ${cloudWatchQuery.results.length}`,
);
```

Ini adalah kelas yang membagi kueri menjadi beberapa langkah jika perlu.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  StartQueryCommand,
  GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}
```

```
export class CloudWatchQuery {
  /**
   * Run a query for all CloudWatch Logs within a certain date range.
   * CloudWatch logs return a max of 10,000 results. This class
   * performs a binary search across all of the logs in the provided
   * date range if a query returns the maximum number of results.
   *
   * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient} client
   * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
   { limit: number } }} config
   */
  constructor(client, { logGroupNames, dateRange, queryConfig }) {
    this.client = client;
    /**
     * All log groups are queried.
     */
    this.logGroupNames = logGroupNames;

    /**
     * The inclusive date range that is queried.
     */
    this.dateRange = dateRange;

    /**
     * CloudWatch Logs never returns more than 10,000 logs.
     */
    this.limit = queryConfig?.limit ?? 10000;

    /**
     * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
     */
    this.results = [];
  }

  /**
   * Run the query.
   */
  async run() {
    this.secondsElapsed = 0;
    const start = new Date();
    this.results = await this._largeQuery(this.dateRange);
    const end = new Date();
    this.secondsElapsed = (end - start) / 1000;
  }
}
```

```
    return this.results;
  }

  /**
   * Recursively query for logs.
   * @param {[Date, Date]} dateRange
   * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[][]>}
   */
  async _largeQuery(dateRange) {
    const logs = await this._query(dateRange, this.limit);

    console.log(
      `Query date range: ${dateRange
        .map((d) => d.toISOString())
        .join(" to ")}. Found ${logs.length} logs.`
    );

    if (logs.length < this.limit) {
      return logs;
    }

    const lastLogDate = this._getLastLogDate(logs);
    const offsetLastLogDate = new Date(lastLogDate);
    offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
    const subDateRange = [offsetLastLogDate, dateRange[1]];
    const [r1, r2] = splitDateRange(subDateRange);
    const results = await Promise.all([
      this._largeQuery(r1),
      this._largeQuery(r2),
    ]);
    return [logs, ...results].flat();
  }

  /**
   * Find the most recent log in a list of logs.
   * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
   */
  _getLastLogDate(logs) {
    const timestamps = logs
      .map(
        (log) =>
          log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
      )
      .filter((t) => !!t)
  }
}
```

```
    .map((t) => `${t}Z`)
    .sort();

    if (!timestamps.length) {
      throw new Error("No timestamp found in logs.");
    }

    return new Date(timestamps[timestamps.length - 1]);
  }

  // snippet-start:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]
  /**
   * Simple wrapper for the GetQueryResultsCommand.
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }
  // snippet-end:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]

  /**
   * Starts a query and waits for it to complete.
   * @param {[Date, Date]} dateRange
   * @param {number} maxLogs
   */
  async _query(dateRange, maxLogs) {
    try {
      const { queryId } = await this._startQuery(dateRange, maxLogs);
      const { results } = await this._waitUntilQueryDone(queryId);
      return results ?? [];
    } catch (err) {
      /**
       * This error is thrown when StartQuery returns an error indicating
       * that the query's start or end date occur before the log group was
       * created.
       */
      if (err instanceof DateOutOfBoundsError) {
        return [];
      } else {
        throw err;
      }
    }
  }
}
```

```
// snippet-start:[javascript.v3.cloudwatch-logs.actions.StartQuery]
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
      // before the log group was created.
      throw new DateOutOfBoundsError(message);
    }

    throw err;
  }
}
// snippet-end:[javascript.v3.cloudwatch-logs.actions.StartQuery]

/**
 * Call GetQueryResultsCommand until the query is done.
 * @param {string} queryId
 */
_waitUntilQueryDone(queryId) {
  const getResults = async () => {
    const results = await this._getQueryResults(queryId);
    const queryDone = [
      "Complete",
      "Failed",
      "Cancelled",
    ]
  }
}
```



```
        "Timeout",
        "Unknown",
    ].includes(results.status);

    return { queryDone, results };
};

return retry(
    { intervalInMs: 1000, maxRetries: 60, quiet: true },
    async () => {
        const { queryDone, results } = await getResults();
        if (!queryDone) {
            throw new Error("Query not done.");
        }

        return results;
    },
);
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [GetQueryResults](#)
 - [StartQuery](#)

CodeBuild contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) with CodeBuild.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Membuat proyek

Contoh kode berikut menunjukkan cara membuat CodeBuild proyek.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat proyek.

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
} from "@aws-sdk/client-codebuild";

// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});

  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
      },
    })
  );
}
```

```
        location: buildOutputBucket,
    },
    // Information about the build environment. The combination of "computeType"
and "type" determines the
    // requirements for the environment such as CPU, memory, and disk space.
    environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
compute-types.html
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
    },
    name: projectName,
    // A role ARN with permission to create a CodeBuild project, write to the
artifact location, and write CloudWatch logs.
    serviceRole: roleArn,
    source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be built.
        location: githubUrl,
    },
    },
    })),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   project: {
//     arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
//     artifacts: {
//       encryptionDisabled: false,
//       location: 'xxxxxx-xxxxxx-xxxxxx',

```

```
//      name: 'MyCodeBuilder',
//      namespaceType: 'NONE',
//      packaging: 'NONE',
//      type: 'S3'
//    },
//    badge: { badgeEnabled: false },
//    cache: { type: 'NO_CACHE' },
//    created: 2023-08-18T14:46:48.979Z,
//    encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
//    environment: {
//      computeType: 'BUILD_GENERAL1_SMALL',
//      environmentVariables: [],
//      image: 'aws/codebuild/standard:7.0',
//      imagePullCredentialsType: 'CODEBUILD',
//      privilegedMode: false,
//      type: 'LINUX_CONTAINER'
//    },
//    lastModified: 2023-08-18T14:46:48.979Z,
//    name: 'MyCodeBuilder',
//    projectVisibility: 'PRIVATE',
//    queuedTimeoutInMinutes: 480,
//    serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
//    source: {
//      insecureSsl: false,
//      location: 'https://...',
//      reportBuildStatus: false,
//      type: 'GITHUB'
//    },
//    timeoutInMinutes: 60
//  }
// }
return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateProject](#) di Referensi AWS SDK for JavaScript API.

Contoh Penyedia Identitas Amazon Cognito menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Penyedia Identitas Amazon Cognito.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Amazon Cognito

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});
```

```
const userPoolNames = [];  
  
for await (const page of paginator) {  
  const names = page.UserPools.map((pool) => pool.Name);  
  userPoolNames.push(...names);  
}  
  
console.log("User pool names: ");  
console.log(userPoolNames.join("\n"));  
return userPoolNames;  
};
```

- Untuk detail API, lihat [ListUserPools](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Konfirmasikan pengguna

Contoh kode berikut menunjukkan cara mengonfirmasi pengguna Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const confirmSignUp = ({ clientId, username, code }) => {  
  const client = new CognitoIdentityProviderClient({});  
  
  const command = new ConfirmSignUpCommand({  
    ClientId: clientId,  
    Username: username,  

```

```
        ConfirmationCode: code,
    });

    return client.send(command);
};
```

- Untuk detail API, lihat [ConfirmSignUp](#) di Referensi AWS SDK for JavaScript API.

Konfirmasikan perangkat MFA untuk melacak

Contoh kode berikut menunjukkan cara mengonfirmasi perangkat MFA untuk dilacak oleh Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new ConfirmDeviceCommand({
        DeviceKey: deviceKey,
        AccessToken: accessToken,
        DeviceSecretVerifierConfig: {
            PasswordVerifier: passwordVerifier,
            Salt: salt,
        },
    });

    return client.send(command);
};
```

- Untuk detail API, lihat [ConfirmDevice](#) di Referensi AWS SDK for JavaScript API.

Dapatkan token untuk mengaitkan aplikasi MFA dengan pengguna

Contoh kode berikut menunjukkan cara mendapatkan token untuk mengaitkan aplikasi MFA dengan pengguna Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [AssociateSoftwareToken](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan informasi tentang pengguna

Contoh kode berikut menunjukkan cara mendapatkan informasi tentang pengguna Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});
```



```
const command = new AdminGetUserCommand({
  UserPoolId: userPoolId,
  Username: username,
});

return client.send(command);
};
```

- Untuk detail API, lihat [AdminGetUser](#) di Referensi AWS SDK for JavaScript API.

Daftar pengguna

Contoh kode berikut menunjukkan cara mencantumkan pengguna Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [ListUsers](#) di Referensi AWS SDK for JavaScript API.

Kirim ulang kode konfirmasi

Contoh kode berikut menunjukkan cara mengirim ulang kode konfirmasi Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [ResendConfirmationCode](#) di Referensi AWS SDK for JavaScript API.

Menanggapi tantangan otentikasi SRP

Contoh kode berikut menunjukkan cara menanggapi tantangan otentikasi Amazon Cognito SRP.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
```

```
code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [RespondToAuthChallenge](#) di Referensi AWS SDK for JavaScript API.

Menanggapi tantangan otentikasi

Contoh kode berikut menunjukkan cara menanggapi tantangan otentikasi Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
  username,
  totp,
  session,
}) => {
  const client = new CognitoIdentityProviderClient({});
```

```
const command = new AdminRespondToAuthChallengeCommand({
  ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
  ChallengeResponses: {
    SOFTWARE_TOKEN_MFA_CODE: totp,
    USERNAME: username,
  },
  ClientId: clientId,
  UserPoolId: userPoolId,
  Session: session,
});

return client.send(command);
};
```

- Untuk detail API, lihat [AdminRespondToAuthChallenge](#) di Referensi AWS SDK for JavaScript API.

Mendaftar pengguna

Contoh kode berikut menunjukkan cara mendaftar pengguna dengan Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};
```

```
};
```

- Untuk detail API, lihat [SignUp](#) di Referensi AWS SDK for JavaScript API.

Mulai otentikasi

Contoh kode berikut menunjukkan cara memulai otentikasi dengan Amazon Cognito.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const initiateAuth = ({ username, password, clientId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
    AuthParameters: {
      USERNAME: username,
      PASSWORD: password,
    },
    ClientId: clientId,
  });


  return client.send(command);
};
```

- Untuk detail API, lihat [InitiateAuth](#) di Referensi AWS SDK for JavaScript API.

Mulai otentikasi dengan kredensi administrator

Contoh kode berikut menunjukkan cara memulai otentikasi dengan Amazon Cognito dan kredensi administrator.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });


  return client.send(command);
};
```

- Untuk detail API, lihat [AdminInitiateAuth](#) di Referensi AWS SDK for JavaScript API.

Verifikasi aplikasi MFA dengan pengguna

Contoh kode berikut menunjukkan cara memverifikasi aplikasi MFA dengan pengguna Amazon Cognito.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});
```

```
// The 'Session' is provided in the response to 'AssociateSoftwareToken'.
const session = process.env.SESSION;

if (!session) {
  throw new Error(
    "Missing a valid Session. Did you run 'admin-initiate-auth'?",
  );
}

const command = new VerifySoftwareTokenCommand({
  Session: session,
  UserCode: totp,
});

return client.send(command);
};
```

- Untuk detail API, lihat [VerifySoftwareToken](#) di Referensi AWS SDK for JavaScript API.

Skenario

Mendaftar pengguna dengan kumpulan pengguna yang membutuhkan MFA

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Daftar dan konfirmasi pengguna dengan nama pengguna, kata sandi, dan alamat email.
- Siapkan otentikasi multi-faktor dengan mengaitkan aplikasi MFA dengan pengguna.
- Masuk dengan menggunakan kata sandi dan kode MFA.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Untuk pengalaman terbaik, kloning GitHub repositori dan jalankan contoh ini. Kode berikut merupakan contoh aplikasi contoh lengkap.

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up' command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Signing up.`);
    await signUp({ clientId, username, password, email });
    log(`Signed up. A confirmation email has been sent to: ${email}.`);
    log(`Run 'confirm-sign-up ${username} <code>' to confirm your account.`);
  } catch (err) {
    log(err);
  }
};
```



```
export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the 'confirm-sign-up' command.`,
    );
  }
};
```

```
    }
  };

const confirmSignUpHandler = async (commands) => {
  const [_ , username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Confirming user.`);
    await confirmSignUp({ clientId, username, code });
    log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
in.` ,
    );
  } catch (err) {
    log(err);
  }
};

export { confirmSignUpHandler };

const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};

import qrCode from "qr-code-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utills/util-log.js";
import { adminInitiateAuth } from "../../actions/admin-initiate-auth.js";
```

```
import { associateSoftwareToken } from "../../../../../actions/associate-software-token.js";
import { FILE_USER_POOLS } from "../constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-token' to finish the setup.",
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
```

```
    `Username and password must be provided as arguments to the 'admin-initiate-
auth' command.` ,
  );
}
};

const adminInitiateAuthHandler = async (commands) => {
  const [, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);

    log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    log(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
```

```
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../actions/admin-respond-to-auth-challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "../constants.js";

const verifyUsername = (username) => {
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-respond-to-auth-challenge' command.`
    );
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an argument to the 'admin-respond-to-auth-challenge' command.`
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);
  }
};
```

```
const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
const session = process.env.SESSION;

const { AuthenticationResult } = await adminRespondToAuthChallenge({
  clientId,
  userPoolId,
  username,
  totp,
  session,
});

storeAccessToken(AuthenticationResult.AccessToken);

log("Successfully authenticated.");
} catch (err) {
  log(err);
}
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-software-token' command.`
    );
  }
};

const verifySoftwareTokenHandler = async (commands) => {
  const [, totp] = commands;

  try {
    validateTotp(totp);

    log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    console.log(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });
};
```

```
    return client.send(command);  
};
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

Contoh DynamoDB menggunakan SDK JavaScript for (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan DynamoDB.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo DynamoDB

Contoh kode berikut ini menunjukkan cara untuk mulai menggunakan DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const client = new DynamoDBClient({});

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

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Buat tabel

Contoh kode berikut menunjukkan cara membuat tabel DynamoDB.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const client = new DynamoDBClient({});

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

```
ProvisionedThroughput: {
  ReadCapacityUnits: 1,
  WriteCapacityUnits: 1,
},
TableName: "CUSTOMER_LIST",
StreamSpecification: {
  StreamEnabled: false,
},
});

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for JavaScript API.

Hapus tabel

Contoh kode berikut menunjukkan cara menghapus tabel DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
```

```
    TableName: "DecafCoffees",
  });

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

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for JavaScript API.

Menghapus item dalam tabel

Contoh kode berikut menunjukkan cara menghapus item dari tabel DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [DeleteCommand](#).

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


const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus item dalam tabel.

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

Hapus item dari tabel menggunakan 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 informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan batch item

Contoh kode berikut menunjukkan bagaimana untuk mendapatkan batch item DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [BatchGet](#).

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

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



```
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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


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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan item dari tabel

Contoh kode berikut menunjukkan bagaimana untuk mendapatkan item dari tabel DynamoDB.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [GetCommand](#).

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


const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

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

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan item dari tabel.

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

Dapatkan item dari tabel menggunakan 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 informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for JavaScript API.

Dapatkan informasi tentang tabel

Contoh kode berikut menunjukkan bagaimana untuk mendapatkan informasi tentang tabel DynamoDB.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const client = new DynamoDBClient({});

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).

- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for JavaScript API.

Mencantumkan tabel

Contoh kode berikut menunjukkan bagaimana daftar tabel DynamoDB.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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


const client = new DynamoDBClient({});

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

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

// Call DynamoDB to retrieve the list of tables
```

```
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for JavaScript API.

Menempatkan item dalam tabel

Contoh kode berikut menunjukkan bagaimana menempatkan item dalam tabel DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [PutCommand](#).

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });
};
```



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

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Masukkan item ke dalam tabel.

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

Masukkan item ke dalam tabel menggunakan 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 informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for JavaScript API.

Mengkueri Tabel

Contoh kode berikut menunjukkan bagaimana untuk query tabel DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [QueryCommand](#).

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for JavaScript .

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for JavaScript .

Menjalankan pernyataan PartiQL

Contoh kode berikut menunjukkan bagaimana menjalankan pernyataan PartiQL pada tabel DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh selengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat item menggunakan PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

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

Dapatkan item menggunakan PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

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

```
    return response;
  };
```

Perbarui item menggunakan PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

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

Hapus item menggunakan PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
```

```
    Parameters: ["Purple"],
  });

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

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for JavaScript API.

Jalankan batch pernyataan PartiQL

Contoh kode berikut menunjukkan bagaimana menjalankan batch pernyataan PartiQL pada tabel DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat batch item menggunakan PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
    })),
  });
```

```
        Parameters: [food],
      })),
    });

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

Dapatkan batch item menggunakan PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

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


Perbarui batch item menggunakan PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

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

Hapus batch item menggunakan PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
```

```
const command = new BatchExecuteStatementCommand({
  Statements: [
    {
      Statement: "DELETE FROM Flavors where Name=?",
      Parameters: ["Grape"],
    },
    {
      Statement: "DELETE FROM Flavors where Name=?",
      Parameters: ["Strawberry"],
    },
  ],
});

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

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for JavaScript API.

Memindai tabel

Contoh kode berikut menunjukkan bagaimana untuk memindai tabel DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [ScanCommand](#).

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};

```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for JavaScript .

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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

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

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
  // want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
};

```

```
// Set the projection expression, which are the attributes that you want.
ProjectionExpression: "Season, Episode, Title, Subtitle",
TableName: "EPISODES_TABLE",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for JavaScript .

Memperbarui item dalam tabel

Contoh kode berikut menunjukkan cara memperbarui item dalam tabel DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [UpdateCommand](#).

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

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

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

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for JavaScript API.

Menulis sekumpulan item

Contoh kode berikut menunjukkan bagaimana menulis batch item DynamoDB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [BatchWrite](#).

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

```
    BatchWriteCommand,
    DynamoDBDocumentClient,
  } from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year' is
        recommended
        // to account for duplicate titles.
        ["BatchWriteMoviesTable"]: putRequests,
      },
    });
  }
};
```

```
    await docClient.send(command);
  }
};
```

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
  {
    PutRequest: {
      Item: {
        KEY: { N: "KEY_VALUE" },
        ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
        ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
      },
    },
  },
}
```

```
    },  
  ],  
},  
};  
  
ddb.batchWriteItem(params, function (err, data) {  
  if (err) {  
    console.log("Error", err);  
  } else {  
    console.log("Success", data);  
  }  
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for JavaScript API.

Skenario

Memulai tabel, item, dan kueri

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat tabel yang dapat menyimpan data film.
- Masukkan, dapatkan, dan perbarui satu film dalam tabel tersebut.
- Tulis data film ke tabel dari file JSON sampel.
- Kueri untuk film yang dirilis pada tahun tertentu.
- Pindai film yang dirilis dalam suatu rentang tahun.
- Hapus film dari tabel, lalu hapus tabel tersebut.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).


```
import { readFileSync } from "fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */
}
```

```
const createTableCommand = new CreateTableCommand({
  TableName: tableName,
  // This example performs a large write to the database.
  // Set the billing mode to PAY_PER_REQUEST to
  // avoid throttling the large write.
  BillingMode: BillingMode.PAY_PER_REQUEST,
  // Define the attributes that are necessary for the key schema.
  AttributeDefinitions: [
    {
      AttributeName: "year",
      // 'N' is a data type descriptor that represents a number type.
      // For a list of all data type descriptors, see the following link.
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
      AttributeType: "N",
    },
    { AttributeName: "title", AttributeType: "S" },
  ],
  // The KeySchema defines the primary key. The primary key can be
  // a partition key, or a combination of a partition key and a sort key.
  // Key schema design is important. For more info, see
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
  KeySchema: [
    // The way your data is accessed determines how you structure your keys.
    // The movies table will be queried for movies by year. It makes sense
    // to make year our partition (HASH) key.
    { AttributeName: "year", KeyType: "HASH" },
    { AttributeName: "title", KeyType: "RANGE" },
  ],
});

log("Creating a table.");
const createTableResponse = await client.send(createTableCommand);
log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

// This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
// You can't write to a table before it's active.
log("Waiting for the table to be active.");
await waitUntilTableExists({ client }, { TableName: tableName });
log("Table active.");

/**
```

```
* Add a movie to the table.
*/

log("Adding a single movie to the table.");
// PutCommand is the first example usage of 'lib-dynamodb'.
const putCommand = new PutCommand({
  TableName: tableName,
  Item: {
    // In 'client-dynamodb', the AttributeValue would be required ( `year: { N:
1981 }` )
    // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
    year: 1981,
    // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
    // is required.
    title: "The Evil Dead",
    // Every other attribute is optional.
    info: {
      genres: ["Horror"],
    },
  },
});
await docClient.send(putCommand);
log("The movie was added.");

/**
 * Get a movie from the table.
 */

log("Getting a single movie from the table.");
const getCommand = new GetCommand({
  TableName: tableName,
  // Requires the complete primary key. For the movies table, the primary key
  // is only the id (partition key).
  Key: {
    year: 1981,
    title: "The Evil Dead",
  },
  // Set this to make sure that recent writes are reflected.
  // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html.
  ConsistentRead: true,
});
const getResponse = await docClient.send(getCommand);
log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);
```

```
/**
 * Update a movie in the table.
 */

log("Updating a single movie in the table.");
const updateCommand = new UpdateCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
  // This update expression appends "Comedy" to the list of genres.
  // For more information on update expressions, see
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
  Expressions.UpdateExpressions.html
  UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
  ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
  ExpressionAttributeValues: {
    ":vals": ["Comedy"],
  },
  ReturnValues: "ALL_NEW",
});
const updateResponse = await docClient.send(updateCommand);
log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

/**
 * Delete a movie from the table.
 */

log("Deleting a single movie from the table.");
const deleteCommand = new DeleteCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
});
await client.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
```

```
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
}
log("Movies added.");

/**
 * Query for movies by year.
 */

log("Querying for all movies from 1981.");
const paginatedQuery = paginateQuery(
  { client: docClient },
  {
    TableName: tableName,
    //For more information about query expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
    KeyConditionExpression: "#y = :y",
    // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
    // name by using an expression attribute name.
    ExpressionAttributeNames: { "#y": "year" },
    ExpressionAttributeValues: { ":y": 1981 },
    ConsistentRead: true,
  },
);
/**
 * @type { Record<string, any>[] };
 */
```

```
const movies1981 = [];
for await (const page of paginatedQuery) {
  movies1981.push(...page.Items);
}
log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

/**
 * Scan the table for movies between 1980 and 1990.
 */

log(`Scan for movies released between 1980 and 1990`);
// A 'Scan' operation always reads every item in the table. If your design
requires
// the use of 'Scan', consider indexing your table or changing your design.
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
const paginatedScan = paginateScan(
  { client: docClient },
  {
    TableName: tableName,
    // Scan uses a filter expression instead of a key condition expression. Scan
will
    // read the entire table and then apply the filter.
    FilterExpression: "#y between :y1 and :y2",
    ExpressionAttributeNames: { "#y": "year" },
    ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
    ConsistentRead: true,
  },
);
/**
 * @type { Record<string, any>[] };
 */
const movies1980to1990 = [];
for await (const page of paginatedScan) {
  movies1980to1990.push(...page.Items);
}
log(
  `Movies: ${movies1980to1990
    .map((m) => `${m.title} (${m.year})`)
    .join(", ")}`,
);

/**
 * Delete the table.
```

```
*/  
  
const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });  
log(`Deleting table ${tableName}.`);  
await client.send(deleteTableCommand);  
log("Table deleted.");  
};
```


- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

Melakukan kueri pada tabel menggunakan batch pernyataan PartiQL

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Dapatkan batch item dengan menjalankan beberapa pernyataan SELECT.
- Tambahkan batch item dengan menjalankan beberapa pernyataan INSERT.
- Perbarui batch item dengan menjalankan beberapa pernyataan UPDATE.
- Hapus batch item dengan menjalankan beberapa pernyataan DELETE.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan pernyataan PartiQL batch.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async () => {
  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
```



```
    AttributeName: "name",
    // 'S' is a data type descriptor that represents a number type.
    // For a list of all data type descriptors, see the following link.
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeType: "S",
  },
],
// The KeySchema defines the primary key. The primary key can be
// a partition key, or a combination of a partition key and a sort key.
// Key schema design is important. For more info, see
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
  KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
});
await client.send(createTableCommand);
log(`Table created: ${tableName}.`);

/**
 * Wait until the table is active.
 */

// This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
// You can't write to a table before it's active.
log("Waiting for the table to be active.");
await waitUntilTableExists({ client }, { TableName: tableName });
log("Table active.");

/**
 * Insert items.
 */

log("Inserting cities into the table.");
const addItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/q1-
reference.insert.html
  Statements: [
    {
      Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
      Parameters: ["Alachua", 10712],
    },
    {
      Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
      Parameters: ["High Springs", 6415],
    }
  ]
});
```

```
    },
  ],
});
await docClient.send(addItemsStatementCommand);
log(`Cities inserted.`);

/**
 * Select items.
 */

log("Selecting cities from the table.");
const selectItemsStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
  Statements: [
    {
      Statement: `SELECT * FROM ${tableName} WHERE name=?`,
      Parameters: ["Alachua"],
    },
    {
      Statement: `SELECT * FROM ${tableName} WHERE name=?`,
      Parameters: ["High Springs"],
    },
  ],
});
const selectItemResponse = await docClient.send(selectItemsStatementCommand);
log(
  `Got cities: ${selectItemResponse.Responses.map(
    (r) => `${r.Item.name} (${r.Item.population})`,
  )}.join(", ")`,
);

/**
 * Update items.
 */

log("Modifying the populations.");
const updateItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
  Statements: [
    {
      Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
      Parameters: [10, "Alachua"],
```

```
    },
    {
      Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
      Parameters: [5, "High Springs"],
    },
  ],
});
await docClient.send(updateItemStatementCommand);
log(`Updated cities.`);

/**
 * Delete the items.
 */

log("Deleting the cities.");
const deleteItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
  Statements: [
    {
      Statement: `DELETE FROM ${tableName} WHERE name=?`,
      Parameters: ["Alachua"],
    },
    {
      Statement: `DELETE FROM ${tableName} WHERE name=?`,
      Parameters: ["High Springs"],
    },
  ],
});
await docClient.send(deleteItemStatementCommand);
log("Cities deleted.");

/**
 * Delete the table.
 */

log("Deleting the table.");
const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
await client.send(deleteTableCommand);
log("Table deleted.");
};
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for JavaScript API.

Melakukan kueri tabel menggunakan PartiQL

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Dapatkan item dengan menjalankan pernyataan SELECT.
- Tambahkan item dengan menjalankan pernyataan INSERT.
- Perbarui item dengan menjalankan pernyataan UPDATE.
- Hapus item dengan menjalankan pernyataan DELETE.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan pernyataan PartiQL tunggal.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async () => {
  /**
   * Create a table.
   */
```

```
log("Creating a table.");
const createTableCommand = new CreateTableCommand({
  TableName: tableName,
  // This example performs a large write to the database.
  // Set the billing mode to PAY_PER_REQUEST to
  // avoid throttling the large write.
  BillingMode: BillingMode.PAY_PER_REQUEST,
  // Define the attributes that are necessary for the key schema.
  AttributeDefinitions: [
    {
      AttributeName: "varietal",
      // 'S' is a data type descriptor that represents a number type.
      // For a list of all data type descriptors, see the following link.
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
      AttributeType: "S",
    },
  ],
  // The KeySchema defines the primary key. The primary key can be
  // a partition key, or a combination of a partition key and a sort key.
  // Key schema design is important. For more info, see
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
  KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
});
await client.send(createTableCommand);
log(`Table created: ${tableName}.`);

/**
 * Wait until the table is active.
 */

// This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
// You can't write to a table before it's active.
log("Waiting for the table to be active.");
await waitUntilTableExists({ client }, { TableName: tableName });
log("Table active.");

/**
 * Insert an item.
 */

log("Inserting a coffee into the table.");
```

```
const addItemStatementCommand = new ExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
  Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
  Parameters: ["arabica", ["chocolate", "floral"]],
});
await client.send(addItemStatementCommand);
log(`Coffee inserted.`);

/**
 * Select an item.
 */

log("Selecting the coffee from the table.");
const selectItemStatementCommand = new ExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
  Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
  Parameters: ["arabica"],
});
const selectItemResponse = await docClient.send(selectItemStatementCommand);
log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

/**
 * Update the item.
 */

log("Add a flavor profile to the coffee.");
const updateItemStatementCommand = new ExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
  Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
varietal=?`,
  Parameters: [["fruity"], "arabica"],
});
await client.send(updateItemStatementCommand);
log(`Updated coffee`);

/**
 * Delete the item.
 */

log("Deleting the coffee.");
const deleteItemStatementCommand = new ExecuteStatementCommand({
```

```
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/q1-  
reference.delete.html  
Statement: `DELETE FROM ${tableName} WHERE varietal=?`,  
Parameters: ["arabica"],  
});  
await docClient.send(deleteItemStatementCommand);  
log("Coffee deleted.");  
  
/**  
 * Delete the table.  
 */  
  
log("Deleting the table.");  
const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });  
await client.send(deleteTableCommand);  
log("Table deleted.");  
};
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for JavaScript API.

Contoh Amazon EC2 menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon EC2.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Ayo Memulai

Halo Amazon EC2

Contoh kode berikut ini menunjukkan cara mendapatkan data tentang tipe instans Amazon EC2.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeSecurityGroupsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  try {
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({}),
    );

    const securityGroupList = SecurityGroups.slice(0, 9)
      .map((sg) => ` • ${sg.GroupId}: ${sg.GroupName}`)
      .join("\n");

    console.log(
      "Hello, Amazon EC2! Let's list up to 10 of your security groups:",
    );
    console.log(securityGroupList);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DescribeSecurityGroups](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Mengalokasikan alamat IP Elastis

Contoh kode berikut menunjukkan cara mengalokasikan alamat IP Elastis untuk Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { AllocateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

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

  try {
    const { AllocationId, PublicIp } = await client.send(command);
    console.log("A new IP address has been allocated to your account:");
    console.log(`ID: ${AllocationId} Public IP: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2. Look under Network & Security > Elastic IPs",
    );
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [AllocateAddress](#) di Referensi AWS SDK for JavaScript API.

Mengaitkan alamat IP Elastis dengan instans

Contoh kode berikut menunjukkan cara mengaitkan alamat IP Elastis dengan instans Amazon EC2.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { AssociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  // You need to allocate an Elastic IP address before associating it with an
  // instance.
  // You can do that with the AllocateAddressCommand.
  const allocationId = "ALLOCATION_ID";
  // You need to create an EC2 instance before an IP address can be associated with
  // it.
  // You can do that with the RunInstancesCommand.
  const instanceId = "INSTANCE_ID";
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  try {
    const { AssociationId } = await client.send(command);
    console.log(
      `Address with allocation ID ${allocationId} is now associated with instance
      ${instanceId}.`,
      `The association ID is ${AssociationId}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [AssociateAddress](#) di Referensi AWS SDK for JavaScript API.

Membuat templat peluncuran

Contoh kode berikut menunjukkan cara membuat template peluncuran Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const ssmClient = new SSMClient({});
const { Parameter } = await ssmClient.send(
  new GetParameterCommand({
    Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
  }),
);
const ec2Client = new EC2Client({});
await ec2Client.send(
  new CreateLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
      UserData: readFileSync(
        join(RESOURCES_PATH, "server_startup_script.sh"),
      ).toString("base64"),
      KeyName: NAMES.keyPairName,
    },
  }),
);
```

- Untuk detail API, lihat [CreateLaunchTemplate](#) di Referensi AWS SDK for JavaScript API.

Membuat grup keamanan

Contoh kode berikut menunjukkan cara membuat grup keamanan Amazon EC2.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateSecurityGroupCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: "SECURITY_GROUP_NAME",
    // Up to 255 characters in length.
    Description: "DESCRIPTION",
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [CreateSecurityGroup](#) di Referensi AWS SDK for JavaScript API.

Membuat pasangan kunci

Contoh kode berikut menunjukkan cara membuat security key pair untuk Amazon EC2.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a key pair in Amazon EC2.
    const { KeyMaterial, KeyName } = await client.send(
      // A unique name for the key pair. Up to 255 ASCII characters.
      new CreateKeyPairCommand({ KeyName: "KEY_PAIR_NAME" }),
    );
    // This logs your private key. Be sure to save it.
    console.log(KeyName);
    console.log(KeyMaterial);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [CreateKeyPair](#) di Referensi AWS SDK for JavaScript API.

Membuat dan menjalankan instans

Contoh kode berikut menunjukkan cara membuat dan menjalankan instans Amazon EC2.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { RunInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Create a new EC2 instance.
export const main = async () => {
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: "KEY_PAIR_NAME",
    // Your security group.
    SecurityGroupIds: ["SECURITY_GROUP_ID"],
    // An x86_64 compatible image.
    ImageId: "ami-0001a0d1a04bfcc30",
    // An x86_64 compatible free-tier instance type.
    InstanceType: "t1.micro",
    // Ensure only 1 instance launches.
    MinCount: 1,
    MaxCount: 1,
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [RunInstances](#) di Referensi AWS SDK for JavaScript API.

Hapus templat peluncuran

Contoh kode berikut menunjukkan cara menghapus template peluncuran Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
await client.send(  
  new DeleteLaunchTemplateCommand({  
    LaunchTemplateName: NAMES.launchTemplateName,  
  }),  
);
```

- Untuk detail API, lihat [DeleteLaunchTemplate](#) di Referensi AWS SDK for JavaScript API.

Menghapus grup keamanan

Contoh kode berikut menunjukkan cara menghapus grup keamanan Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteSecurityGroupCommand } from "@aws-sdk/client-ec2";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  const command = new DeleteSecurityGroupCommand({  
    GroupId: "GROUP_ID",  
  });  
  
  try {  
    await client.send(command);  
    console.log("Security group deleted successfully.");  
  } catch (err) {  
    console.error(err);  
  }  
};
```

- Untuk detail API, lihat [DeleteSecurityGroup](#) di Referensi AWS SDK for JavaScript API.

Membuat pasangan kunci keamanan

Contoh kode berikut menunjukkan cara menghapus key pair keamanan Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DeleteKeyPairCommand({
    KeyName: "KEY_PAIR_NAME",
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DeleteKeyPair](#) di Referensi AWS SDK for JavaScript API.

Menjelaskan Wilayah

Contoh kode berikut menunjukkan cara mendeskripsikan Wilayah Amazon EC2.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeRegionsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions true even the regions that require opt-in will be returned.
    AllRegions: true,
    // You can omit the Filters property if you want to get all regions.
    Filters: [
      {
        Name: "region-name",
        // You can specify multiple values for a filter.
        // You can also use '*' as a wildcard. This will return all
        // of the regions that start with `us-east-`.
        Values: ["ap-southeast-4"],
      },
    ],
  });

  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DescribeRegions](#) di Referensi AWS SDK for JavaScript API.

Menjelaskan instans

Contoh kode berikut menunjukkan cara mendeskripsikan instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List all of your EC2 instances running with x86_64 architecture that were
// launched this month.
export const main = async () => {
  const d = new Date();
  const year = d.getFullYear();
  const month = `${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;
  const command = new DescribeInstancesCommand({
    Filters: [
      { Name: "architecture", Values: ["x86_64"] },
      { Name: "instance-state-name", Values: ["running"] },
      {
        Name: "launch-time",
        Values: [launchTimePattern],
      },
    ],
  });

  try {
    const { Reservations } = await client.send(command);
    const instanceList = Reservations.reduce((prev, current) => {
      return prev.concat(current.Instances);
    }, []);

    console.log(instanceList);
  } catch (err) {
    console.error(err);
  }
};
```

```
  }  
};
```

- Untuk detail API, lihat [DescribeInstances](#) di Referensi AWS SDK for JavaScript API.

Menonaktifkan pemantauan terperinci

Contoh kode berikut menunjukkan cara menonaktifkan pemantauan terperinci pada instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  const command = new UnmonitorInstancesCommand({  
    InstanceIds: ["i-09a3dfe7ae00e853f"],  
  });  
  
  try {  
    const { InstanceMonitorings } = await client.send(command);  
    const instanceMonitoringsList = InstanceMonitorings.map(  
      (im) =>  
        ` • Detailed monitoring state for ${im.InstanceId} is  
${im.Monitoring.State}.`,  
    );  
    console.log("Monitoring status:");  
    console.log(instanceMonitoringsList.join("\n"));  
  } catch (err) {  
    console.error(err);  
  }  
};
```

- Untuk detail API, lihat [UnmonitorInstances](#) di Referensi AWS SDK for JavaScript API.

Memisahkan alamat IP Elastis dari instans

Contoh kode berikut menunjukkan cara memisahkan alamat IP Elastis dari instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DisassociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Disassociate an Elastic IP address from an instance.
export const main = async () => {
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
    // retired.
    AssociationId: "ASSOCIATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DisassociateAddress](#) di Referensi AWS SDK for JavaScript API.

Mengaktifkan pemantauan

Contoh kode berikut menunjukkan cara mengaktifkan pemantauan untuk instans Amazon EC2 yang sedang berjalan.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { MonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Turn on detailed monitoring for the selected instance.
// By default, metrics are sent to Amazon CloudWatch every 5 minutes.
// For a cost you can enable detailed monitoring which sends metrics every minute.
export const main = async () => {
  const command = new MonitorInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instancesBeingMonitored = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instancesBeingMonitored.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [MonitorInstances](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan data tentang Amazon Machine Images

Contoh kode berikut menunjukkan cara mendapatkan data tentang Amazon Machine Images (AMI).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { paginateDescribeImages } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first i386 image available for EC2 instances.
export const main = async () => {
  // The paginate function is a wrapper around the base command.
  const paginator = paginateDescribeImages(
    // Without limiting the page size, this call can take a long time. pageSize is
    // just sugar for
    // the MaxResults property in the base command.
    { client, pageSize: 25 },
    {
      // There are almost 70,000 images available. Be specific with your filtering
      // to increase efficiency.
      // See https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
      // ec2/interfaces/describeimagescommandinput.html#filters
      Filters: [{ Name: "architecture", Values: ["x86_64"] }],
    },
  );

  try {
    const arm64Images = [];
    for await (const page of paginator) {
      if (page.Images.length) {
        arm64Images.push(...page.Images);
        // Once we have at least 1 result, we can stop.
        if (arm64Images.length >= 1) {
          break;
        }
      }
    }
  }
}
```

```
    }
    console.log(arm64Images);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DescribeImages](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan data tentang grup keamanan

Contoh kode berikut menunjukkan cara mendapatkan data tentang grup keamanan Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeSecurityGroupsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Log the details of a specific security group.
export const main = async () => {
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: ["SECURITY_GROUP_ID"],
  });

  try {
    const { SecurityGroups } = await client.send(command);
    console.log(JSON.stringify(SecurityGroups, null, 2));
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DescribeSecurityGroups](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan data tentang tipe instans

Contoh kode berikut menunjukkan cara mendapatkan data tentang jenis instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import {
  paginateDescribeInstanceTypes,
  DescribeInstanceTypesCommand,
} from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first arm64 EC2 instance type available.
export const main = async () => {
  // The paginate function is a wrapper around the underlying command.
  const paginator = paginateDescribeInstanceTypes(
    // Without limiting the page size, this call can take a long time. pageSize is
    just sugar for
    // the MaxResults property in the underlying command.
    { client, pageSize: 25 },
    {
      Filters: [
        { Name: "processor-info.supported-architecture", Values: ["x86_64"] },
        { Name: "free-tier-eligible", Values: ["true"] },
      ],
    }
  );

  try {
    const instanceTypes = [];

    for await (const page of paginator) {
      if (page.InstanceTypes.length) {
```



```
instanceTypes.push(...page.InstanceTypes);

// When we have at least 1 result, we can stop.
if (instanceTypes.length >= 1) {
  break;
}
}
}
console.log(instanceTypes);
} catch (err) {
  console.error(err);
}
};
```

- Untuk detail API, lihat [DescribeInstanceTypes](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan data tentang profil instans yang terkait dengan instans

Contoh kode berikut menunjukkan cara mendapatkan data tentang profil instans yang terkait dengan instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
```

- Untuk detail API, lihat [DescribeInstanceProfileAssociations](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan detail tentang alamat IP Elastis

Contoh kode berikut menunjukkan cara mendapatkan detail tentang alamat IP Elastis.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeAddressesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: ["ALLOCATION_ID"],
  });

  try {
    const { Addresses } = await client.send(command);
    const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
    console.log("Elastic IP addresses:");
    console.log(addressList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DescribeAddresses](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan VPC default

Contoh kode berikut menunjukkan cara mendapatkan VPC default dari akun saat ini.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```

- Untuk detail API, lihat [DescribeVpcs](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan subnet default untuk VPC

Contoh kode berikut menunjukkan cara mendapatkan subnet default untuk VPC.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  }),
);
```

- Untuk detail API, lihat [DescribeSubnets](#) di Referensi AWS SDK for JavaScript API.

Membuat daftar pasangan kunci keamanan

Contoh kode berikut menunjukkan cara membuat daftar pasangan kunci keamanan Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeKeyPairsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

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

  try {
    const { KeyPairs } = await client.send(command);
    const keyPairList = KeyPairs.map(
      (kp) => ` • ${kp.KeyPairId}: ${kp.KeyName}`,
    ).join("\n");
    console.log("The following key pairs were found in your account:");
    console.log(keyPairList);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DescribeKeyPairs](#) di Referensi AWS SDK for JavaScript API.

Menyalakan ulang instans

Contoh kode berikut menunjukkan cara me-reboot instans Amazon EC2.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { RebootInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new RebootInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [RebootInstances](#) di Referensi AWS SDK for JavaScript API.

Merilis alamat IP Elastis

Contoh kode berikut menunjukkan cara melepaskan alamat IP Elastis.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { ReleaseAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
    // retired.
    AllocationId: "ALLOCATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully released address.");
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [ReleaseAddress](#) di Referensi AWS SDK for JavaScript API.

Mengganti profil instans yang terkait dengan instans

Contoh kode berikut menunjukkan cara mengganti profil instans yang terkait dengan instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
```


```
    ),  
  );
```

- Untuk detail API, lihat [ReplacelamInstanceProfileAssociation](#) di Referensi AWS SDK for JavaScript API.

Menetapkan aturan masuk untuk grup keamanan

Contoh kode berikut menunjukkan cara menetapkan aturan masuk untuk grup keamanan Amazon EC2.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { AuthorizeSecurityGroupIngressCommand } from "@aws-sdk/client-ec2";  
  
import { client } from "../libs/client.js";  
  
// Grant permissions for a single IP address to ssh into instances  
// within the provided security group.  
export const main = async () => {  
  const command = new AuthorizeSecurityGroupIngressCommand({  
    // Replace with a security group ID from the AWS console or  
    // the DescribeSecurityGroupsCommand.  
    GroupId: "SECURITY_GROUP_ID",  
    IpPermissions: [  
      {  
        IpProtocol: "tcp",  
        FromPort: 22,  
        ToPort: 22,  
        // Replace 0.0.0.0 with the IP address to authorize.  
        // For more information on this notation, see  
        // https://en.wikipedia.org/wiki/Classless_Inter-  
        Domain_Routing#CIDR_notation  
        IpRanges: [{ CidrIp: "0.0.0.0/32" }],  
      }  
    ]  
  });
```

```
    },
  ],
});

try {
  const { SecurityGroupRules } = await client.send(command);
  console.log(JSON.stringify(SecurityGroupRules, null, 2));
} catch (err) {
  console.error(err);
}
};
```

- Untuk detail API, lihat [AuthorizeSecurityGroupIngress](#) di Referensi AWS SDK for JavaScript API.

Memulai instans

Contoh kode berikut menunjukkan cara memulai instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { StartInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new StartInstancesCommand({
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
  }
};
```



```
);
console.log("Starting instances:");
console.log(instanceIdList.join("\n"));
} catch (err) {
  console.error(err);
}
};
```

- Untuk detail API, lihat [StartInstances](#) di Referensi AWS SDK for JavaScript API.

Menghentikan instans

Contoh kode berikut menunjukkan cara menghentikan instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { StopInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new StopInstancesCommand({
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

```
}  
};
```

- Untuk detail API, lihat [StopInstances](#) di Referensi AWS SDK for JavaScript API.

Akhiri instans

Contoh kode berikut menunjukkan cara menghentikan instans Amazon EC2.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { TerminateInstancesCommand } from "@aws-sdk/client-ec2";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  const command = new TerminateInstancesCommand({  
    InstanceIds: ["INSTANCE_ID"],  
  });  
  
  try {  
    const { TerminatingInstances } = await client.send(command);  
    const instanceList = TerminatingInstances.map(  
      (instance) => ` • ${instance.InstanceId}`,  
    );  
    console.log("Terminating instances:");  
    console.log(instanceList.join("\n"));  
  } catch (err) {  
    console.error(err);  
  }  
};
```

- Untuk detail API, lihat [TerminateInstances](#) di Referensi AWS SDK for JavaScript API.

Skenario

Membangun dan mengelola layanan yang tangguh

Contoh kode berikut menunjukkan cara membuat layanan web load-balanced yang mengembalikan rekomendasi buku, film, dan lagu. Contoh ini menunjukkan cara layanan tersebut merespons kegagalan, serta cara merestrukturisasi layanan agar lebih tangguh ketika terjadi kegagalan.

- Menggunakan grup Amazon EC2 Auto Scaling untuk membuat instans Amazon Elastic Compute Cloud (Amazon EC2) berdasarkan templat peluncuran dan menyimpan sejumlah instans dalam rentang yang ditentukan.
- Menangani dan mendistribusikan permintaan HTTP dengan Elastic Load Balancing.
- Memantau kondisi instans dalam grup Auto Scaling dan meneruskan permintaan hanya ke instans yang sehat.
- Menjalankan server web Python pada setiap instans EC2 untuk menangani permintaan HTTP. Server web merespons dengan memberikan rekomendasi dan melakukan pemeriksaan kondisi.
- Menyimulasikan layanan yang direkomendasikan dengan tabel Amazon DynamoDB.
- Kontrol respons server web terhadap permintaan dan pemeriksaan kesehatan dengan memperbarui AWS Systems Manager parameter.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menjalankan skenario interaktif di prompt perintah.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
```

```
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 * - deploy
 * - demo
 * - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

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

Menyusun langkah-langkah untuk men-deploy semua sumber daya.

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

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
```

```
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
```

```
        AttributeType: "N",
      },
    ],
    KeySchema: [
      {
        AttributeName: "MediaType",
        KeyType: "HASH",
      },
      {
        AttributeName: "ItemId",
        KeyType: "RANGE",
      },
    ],
  })),
);
await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
  "createdTable",
  MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "populatingTable",
  MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
}),
```

```
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
  const { KeyMaterial } = await client.send(
    new CreateKeyPairCommand({
      KeyName: NAMES.keyPairName,
    }),
  );

  writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  );
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
```



```
MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
    ),
),
new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
        new CreateRoleCommand({
            RoleName: NAMES.instanceRoleName,
            AssumeRolePolicyDocument: readFileSync(
                join(ROOT, "assume-role-policy.json"),
            ),
        }),
    );
}),
new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
    ),
),
new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
        new AttachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: state.instancePolicyArn,
        }),
    );
}),
}),
```

```
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
```

```

        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
    })),
    );
  })),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    ),
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
  })),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),

```

```

    ),
    new ScenarioOutput(
      "creatingAutoScalingGroup",
      MESSAGES.creatingAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      ),
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {

```

```

    type: "confirm",
  )),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
  )),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  )),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",

```

```
MESSAGES.creatingLoadBalancerTargetGroup.replace(
  "${TARGET_GROUP_NAME}",
  NAMES.loadBalancerTargetGroupName,
),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
  state.targetGroupProtocol = targetGroup.Protocol;
  state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
  "createdLoadBalancerTargetGroup",
  MESSAGES.createdLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
```

```
        Subnets: state.subnets,
    })),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
        { client },
        { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
),
new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
        new CreateListenerCommand({
            LoadBalancerArn: state.loadBalancerArn,
            Protocol: state.targetGroupProtocol,
            Port: state.targetGroupPort,
            DefaultActions: [
                { Type: "forward", TargetGroupArn: state.targetGroupArn },
            ],
        }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
        "${LB_LISTENER_ARN}",
        state.loadBalancerListenerArn,
    ),
),
```

```

    ),
    new ScenarioOutput(
      "attachingLoadBalancerTargetGroup",
      MESSAGES.attachingLoadBalancerTargetGroup
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
    ),
    new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
      const client = new AutoScalingClient({});
      await client.send(
        new AttachLoadBalancerTargetGroupsCommand({
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          TargetGroupARNs: [state.targetGroupArn],
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    }),
    new ScenarioOutput(
      "attachedLoadBalancerTargetGroup",
      MESSAGES.attachedLoadBalancerTargetGroup,
    ),
    new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
    new ScenarioAction(
      "verifyInboundPort",
      /**
       *
       * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
       */
      async (state) => {
        const client = new EC2Client({});
        const { SecurityGroups } = await client.send(
          new DescribeSecurityGroupsCommand({
            Filters: [{ Name: "group-name", Values: ["default"] }],
          }),
        );
        if (!SecurityGroups) {
          state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
        }
        state.defaultSecurityGroup = SecurityGroups[0];

        /**
         * @type {string}

```



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

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

```
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  })
}
```

```
    }),  
    new ScenarioOutput("verifiedEndpoint", (state) => {  
      if (state.verifyEndpointError) {  
        console.error(state.verifyEndpointError);  
      } else {  
        return MESSAGES.verifiedEndpoint.replace(  
          "${ENDPOINT_RESPONSE}",  
          state.endpointResponse,  
        );  
      }  
    })),  
  ];  
];
```

Menyusun langkah-langkah untuk menjalankan demo.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { readFileSync } from "node:fs";  
import { join } from "node:path";  
  
import axios from "axios";  
  
import {  
  DescribeTargetGroupsCommand,  
  DescribeTargetHealthCommand,  
  ElasticLoadBalancingV2Client,  
} from "@aws-sdk/client-elastic-load-balancing-v2";  
import {  
  DescribeInstanceInformationCommand,  
  PutParameterCommand,  
  SSMClient,  
  SendCommandCommand,  
} from "@aws-sdk/client-ssm";  
import {  
  IAMClient,  
  CreatePolicyCommand,  
  CreateRoleCommand,  
  AttachRolePolicyCommand,  
  CreateInstanceProfileCommand,  
  AddRoleToInstanceProfileCommand,  
  waitUntilInstanceProfileExists,  
} from "@aws-sdk/client-iam";
```

```
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
```

```

    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
    { preformatted: true },
  );

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
  balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {

```

```
    inputEquals: true,
    input: new ScenarioInput(
      "loadBalancerCheck",
      MESSAGES.demoLoadBalancerCheck,
      {
        type: "confirm",
      },
    ),
    output: getRecommendationResult,
  },
},
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
```

```
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        })
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        })
      );
    }
  })
}
```

```

    }),
    new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
    ...statusSteps,
    new ScenarioInput(
      "badCredentialsConfirmation",
      MESSAGES.demoBadCredentialsConfirmation,
      { type: "confirm" },
    ),
    new ScenarioAction("badCredentialsExit", (state) => {
      if (!state.badCredentialsConfirmation) {
        process.exit();
      }
    }),
    new ScenarioAction("fixDynamoDBName", async () => {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: NAMES.tableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }),
    new ScenarioAction(
      "badCredentials",
      /**
       * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
       state
       */
      async (state) => {
        await createSsmOnlyInstanceProfile();
        const autoScalingClient = new AutoScalingClient({});
        const { AutoScalingGroups } = await autoScalingClient.send(
          new DescribeAutoScalingGroupsCommand({
            AutoScalingGroupNames: [NAMES.autoScalingGroupName],
          }),
        );
        state.targetInstance = AutoScalingGroups[0].Instances[0];
        // snippet-start:
        [javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
        const ec2Client = new EC2Client({});
        const { IamInstanceProfileAssociations } = await ec2Client.send(
          new DescribeIamInstanceProfileAssociationsCommand({

```



```
        Filters: [
          { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
        ],
      )),
    );
    // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
    state.instanceProfileAssociationId =
      iamInstanceProfileAssociations[0].AssociationId;
    // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );
    // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

    await ec2Client.send(
      new RebootInstancesCommand({
        InstanceIds: [state.targetInstance.InstanceId],
      }),
    );

    const ssmClient = new SSMClient({});
    await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
      const { InstanceInformationList } = await ssmClient.send(
        new DescribeInstanceInformationCommand({}),
      );

      const instance = InstanceInformationList.find(
        (info) => info.InstanceId === state.targetInstance.InstanceId,
      );

      if (!instance) {
        throw new Error("Instance not found.");
      }
    });

    await ssmClient.send(
```

```

        new SendCommandCommand({
            InstanceIds: [state.targetInstance.InstanceId],
            DocumentName: "AWS-RunShellScript",
            Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
    );
},
),
new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
    */
    (state) =>
        MESSAGES.demoTestBadCredentials.replace(
            "${INSTANCE_ID}",
            state.targetInstance.InstanceId,
        ),
    ),
loadBalancerLoop,
new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
        process.exit();
    }
}),
new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
        new PutParameterCommand({
            Name: NAMES.ssmHealthCheckKey,
            Value: "deep",
            Overwrite: true,
            Type: "String",
        }),
    );
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,

```

```
loadBalancerLoop,
new ScenarioInput(
  "killInstanceConfirmation",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
  ssm').InstanceInformation }} state
   */
  (state) =>
    MESSAGES.demoKillInstanceConfirmation.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
  { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
  if (!state.killInstanceConfirmation) {
    process.exit();
  }
}),
new ScenarioAction(
  "killInstance",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
  ssm').InstanceInformation }} state
   */
  async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new TerminateInstanceInAutoScalingGroupCommand({
        InstanceId: state.targetInstance.InstanceId,
        ShouldDecrementDesiredCapacity: false,
      }),
    );
  },
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {
    process.exit();
  }
});
```

```

    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});

```

```
const { Policy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyName: NAMES.ssmOnlyPolicyName,
    PolicyDocument: readFileSync(
      join(RESOURCES_PATH, "ssm_only_policy.json"),
    ),
  }),
);
await iamClient.send(
  new CreateRoleCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: { Service: "ec2.amazonaws.com" },
          Action: "sts:AssumeRole",
        },
      ],
    }),
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: Policy.Arn,
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
  }),
);
// snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```

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

return InstanceProfile;
}
```

Menyusun langkah-langkah untuk menghancurkan semua sumber daya.

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

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
```

```
    DeleteTargetGroupCommand,
    DescribeTargetGroupsCommand,
    ElasticLoadBalancingV2Client,
  } from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
```

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

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  })
}
```



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

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

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

```
    return MESSAGES.deletedInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  } else {
    return MESSAGES.deletedInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
  try {
    await terminateGroupInstances(NAMES.autoScalingGroupName);
    await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
      await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
    });
  } catch (e) {
    state.deleteAutoScalingGroupError = e;
  }
}),
```

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

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

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
```

```
        client.send(
            new DeleteTargetGroupCommand({
                TargetGroupArn: TargetGroups[0].TargetGroupArn,
            }),
        ),
    );
} catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
}
// snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
        console.error(state.deleteLoadBalancerTargetGroupError);
        return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
            "${TARGET_GROUP_NAME}",
            NAMES.loadBalancerTargetGroupName,
        );
    } else {
        return MESSAGES.deletedLoadBalancerTargetGroup.replace(
            "${TARGET_GROUP_NAME}",
            NAMES.loadBalancerTargetGroupName,
        );
    }
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
        const client = new IAMClient({});
        await client.send(
            new RemoveRoleFromInstanceProfileCommand({
                InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
                RoleName: NAMES.ssmOnlyRoleName,
            }),
        );
    } catch (e) {
        state.detachSsmOnlyRoleFromProfileError = e;
    }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
        console.error(state.detachSsmOnlyRoleFromProfileError);
        return MESSAGES.detachSsmOnlyRoleFromProfileError
            .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
            .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
});
```

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

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



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

```

```
        NAMES.ssmOnlyRoleName,
    );
  }
  })),
];

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

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

/**
 * @param {string} groupName
 */
```

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

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

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for JavaScript .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)

- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacesInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)


Memulai instans

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Membuat pasangan kunci dan grup keamanan.
- Memilih Amazon Machine Image (AMI) dan tipe instans yang kompatibel, lalu membuat instans.
- Menghentikan dan memulai ulang instans.
- Kaitkan alamat IP Elastis dengan instans Anda.

Menghubungkan instans Anda dengan SSH, lalu membersihkan sumber daya.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan skenario interaktif di prompt perintah.

```
import { mkdtempSync, writeFileSync, rmSync } from "fs";
import { tmpdir } from "os";
import { join } from "path";
import { get } from "http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
  DeleteSecurityGroupCommand,
  DescribeInstancesCommand,
  DescribeKeyPairsCommand,
  DescribeSecurityGroupsCommand,
  DisassociateAddressCommand,
  EC2Client,
  paginateDescribeImages,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";
import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
```

```
const ec2Client = new EC2Client();
const ssmClient = new SSMClient();

const prompter = new Prompter();
const confirmMessage = "Continue?";
const tmpDirectory = mkdtempSync(join(tmpdir(), "ec2-scenario-tmp"));

const createKeyPair = async (keyPairName) => {
  // Create a key pair in Amazon EC2.
  const { KeyMaterial, KeyPairId } = await ec2Client.send(
    // A unique name for the key pair. Up to 255 ASCII characters.
    new CreateKeyPairCommand({ KeyName: keyPairName }),
  );

  // Save the private key in a temporary location.
  writeFileSync(`${tmpDirectory}/${keyPairName}.pem`, KeyMaterial, {
    mode: 0o400,
  });

  return KeyPairId;
};

const describeKeyPair = async (keyPairName) => {
  const command = new DescribeKeyPairsCommand({
    KeyNames: [keyPairName],
  });
  const { KeyPairs } = await ec2Client.send(command);
  return KeyPairs[0];
};

const createSecurityGroup = async (securityGroupName) => {
  const command = new CreateSecurityGroupCommand({
    GroupName: securityGroupName,
    Description: "A security group for the Amazon EC2 example.",
  });
  const { GroupId } = await ec2Client.send(command);
  return GroupId;
};

const allocateIpAddress = async () => {
  const command = new AllocateAddressCommand({});
  const { PublicIp, AllocationId } = await ec2Client.send(command);
  return { PublicIp, AllocationId };
};
```

```
};

const getLocalIpAddress = () => {
  return new Promise((res, rej) => {
    get("http://checkip.amazonaws.com", (response) => {
      let data = "";
      response.on("data", (chunk) => (data += chunk));
      response.on("end", () => res(data.trim()));
    }).on("error", (err) => {
      rej(err);
    });
  });
};

const authorizeSecurityGroupIngress = async (securityGroupId) => {
  const ipAddress = await getLocalIpAddress();
  const command = new AuthorizeSecurityGroupIngressCommand({
    GroupId: securityGroupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  await ec2Client.send(command);
  return ipAddress;
};

const describeSecurityGroup = async (securityGroupName) => {
  const command = new DescribeSecurityGroupsCommand({
    GroupNames: [securityGroupName],
  });
  const { SecurityGroups } = await ec2Client.send(command);

  return SecurityGroups[0];
};

const getAmznLinux2AMIs = async () => {
  const AMIs = [];
  for await (const page of paginateGetParametersByPath(
```

```

    {
      client: ssmClient,
    },
    { Path: "/aws/service/ami-amazon-linux-latest" },
  )) {
    page.Parameters.forEach((param) => {
      if (param.Name.includes("amzn2")) {
        AMIs.push(param.Value);
      }
    });
  }

  const imageDetails = [];

  for await (const page of paginateDescribeImages(
    { client: ec2Client },
    { ImageIds: AMIs },
  )) {
    imageDetails.push(...(page.Images || []));
  }

  const choices = imageDetails.map((image, index) => ({
    name: `${image.ImageId} - ${image.Description}`,
    value: index,
  }));

  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an image.",
    choices,
  });

  return imageDetails[selectedIndex];
};

/**
 * @param {import('@aws-sdk/client-ec2').Image} imageDetails
 */
const getCompatibleInstanceTypes = async (imageDetails) => {
  const paginator = paginateDescribeInstanceTypes(
    { client: ec2Client, pageSize: 25 },
    {

```



```
    Filters: [
      {
        Name: "processor-info.supported-architecture",
        Values: [imageDetails.Architecture],
      },
      { Name: "instance-type", Values: ["*.micro", "*.small"] },
    ],
  },
);

const instanceTypes = [];

for await (const page of paginator) {
  if (page.InstanceTypes.length) {
    instanceTypes.push(...(page.InstanceTypes || []));
  }
}

const choices = instanceTypes.map((type, index) => ({
  name: `${type.InstanceType} - Memory:${type.MemoryInfo.SizeInMiB}`,
  value: index,
}));

/**
 * @type {number}
 */
const selectedIndex = await prompter.select({
  message: "Select an instance type.",
  choices,
});
return instanceTypes[selectedIndex];
};

const runInstance = async ({
  keyPairName,
  securityGroupId,
  imageId,
  instanceType,
}) => {
  const command = new RunInstancesCommand({
    KeyName: keyPairName,
    SecurityGroupIds: [securityGroupId],
    ImageId: imageId,
    InstanceType: instanceType,
```

```
    MinCount: 1,
    MaxCount: 1,
  });

  const { Instances } = await ec2Client.send(command);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [Instances[0].InstanceId] },
  );
  return Instances[0].InstanceId;
};

const describeInstance = async (instanceId) => {
  const command = new DescribeInstancesCommand({
    InstanceIds: [instanceId],
  });

  const { Reservations } = await ec2Client.send(command);
  return Reservations[0].Instances[0];
};

const displaySSHConnectionInfo = ({ publicIp, keyPairName }) => {
  return `ssh -i ${tmpDirectory}/${keyPairName}.pem ec2-user@${publicIp}`;
};

const stopInstance = async (instanceId) => {
  const command = new StopInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(command);
  await waitUntilInstanceStopped(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
};

const startInstance = async (instanceId) => {
  const startCommand = new StartInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(startCommand);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
  return await describeInstance(instanceId);
};
```

```
const associateAddress = async ({ allocationId, instanceId }) => {
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  const { AssociationId } = await ec2Client.send(command);
  return AssociationId;
};

const disassociateAddress = async (associationId) => {
  const command = new DisassociateAddressCommand({
    AssociationId: associationId,
  });
  try {
    await ec2Client.send(command);
  } catch (err) {
    console.warn(
      `Failed to disassociated address with association id: ${associationId}`,
      err,
    );
  }
};

const releaseAddress = async (allocationId) => {
  const command = new ReleaseAddressCommand({
    AllocationId: allocationId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Address with allocation ID ${allocationId} released.\n`);
  } catch (err) {
    console.log(
      `Failed to release address with allocation id: ${allocationId}.`,
      err,
    );
  }
};

const restartInstance = async (instanceId) => {
  console.log("Stopping instance.");
  await stopInstance(instanceId);
  console.log("Instance stopped.");
};
```

```
    console.log("Starting instance.");
    const { PublicIpAddress } = await startInstance(instanceId);
    return PublicIpAddress;
  };

const terminateInstance = async (instanceId) => {
  const command = new TerminateInstancesCommand({
    InstanceIds: [instanceId],
  });

  try {
    await ec2Client.send(command);
    await waitUntilInstanceTerminated(
      { client: ec2Client },
      { InstanceIds: [instanceId] },
    );
    console.log(`Instance with ID ${instanceId} terminated.\n`);
  } catch (err) {
    console.warn(`Failed to terminate instance ${instanceId}.`, err);
  }
};

const deleteSecurityGroup = async (securityGroupId) => {
  const command = new DeleteSecurityGroupCommand({
    GroupId: securityGroupId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Security group ${securityGroupId} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete security group ${securityGroupId}.`, err);
  }
};

const deleteKeyPair = async (keyPairName) => {
  const command = new DeleteKeyPairCommand({
    KeyName: keyPairName,
  });

  try {
    await ec2Client.send(command);
    console.log(`Key pair ${keyPairName} deleted.\n`);
  } catch (err) {
```

```
    console.warn(`Failed to delete key pair ${keyPairName}.`, err);
  }
};

const deleteTemporaryDirectory = () => {
  try {
    rmSync(tmpDirectory, { recursive: true });
    console.log(`Temporary directory ${tmpDirectory} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete temporary directory ${tmpDirectory}.`, err);
  }
};

export const main = async () => {
  const keyPairName = "ec2-scenario-key-pair";
  const securityGroupName = "ec2-scenario-security-group";

  let securityGroupId, ipAllocationId, publicIp, instanceId, associationId;

  console.log(wrapText("Welcome to the Amazon EC2 basic usage scenario.));

  try {
    // Prerequisites
    console.log(
      "Before you launch an instance, you'll need a few things:",
      "\n - A Key Pair",
      "\n - A Security Group",
      "\n - An IP Address",
      "\n - An AMI",
      "\n - A compatible instance type",
      "\n\n I'll go ahead and take care of the first three, but I'll need your help
for the rest.",
    );

    await prompter.confirm({ message: confirmMessage });

    await createKeyPair(keyPairName);
    securityGroupId = await createSecurityGroup(securityGroupName);
    const { PublicIp, AllocationId } = await allocateIpAddress();
    ipAllocationId = AllocationId;
    publicIp = PublicIp;
    const ipAddress = await authorizeSecurityGroupIngress(securityGroupId);

    const { KeyName } = await describeKeyPair(keyPairName);
```

```
const { GroupName } = await describeSecurityGroup(securityGroupName);
console.log(`# created the key pair ${KeyName}.\n`);
console.log(
  `# created the security group ${GroupName}`,
  `and allowed SSH access from ${ipAddress} (your IP).\n`,
);
console.log(`# allocated ${publicIp} to be used for your EC2 instance.\n`);

await prompter.confirm({ message: confirmMessage });

// Creating the instance
console.log(wrapText("Create the instance."));
console.log(
  "You get to choose which image you want. Select an amazon-linux-2 image from
the following:",
);
const imageDetails = await getAmznLinux2AMIs();
const instanceTypeDetails = await getCompatibleInstanceTypes(imageDetails);
console.log("Creating your instance. This can take a few seconds.");
instanceId = await runInstance({
  keyPairName,
  securityGroupId,
  imageId: imageDetails.ImageId,
  instanceType: instanceTypeDetails.InstanceType,
});
const instanceDetails = await describeInstance(instanceId);
console.log(`# instance ${instanceId}.\n`);
console.log(instanceDetails);
console.log(
  `\nYou should now be able to SSH into your instance from another terminal:`,
  `\n${displaySSHConnectionInfo({
    publicIp: instanceDetails.PublicIpAddress,
    keyPairName,
  })}`,
);

await prompter.confirm({ message: confirmMessage });

// Understanding the IP address.
console.log(wrapText("Understanding the IP address."));
console.log(
  "When you stop and start an instance, the IP address will change. I'll restart
your",
  "instance for you. Notice how the IP address changes.",
);
```

```
);
const ipAddressAfterRestart = await restartInstance(instanceId);
console.log(
  `
Instance started. The IP address changed from
${instanceDetails.PublicIpAddress} to ${ipAddressAfterRestart}`,
  `
${displaySSHConnectionInfo({
  publicIp: ipAddressAfterRestart,
  keyPairName,
})}`
);
await prompter.confirm({ message: confirmMessage });
console.log(
  `If you want to the IP address to be static, you can associate an allocated`,
  `IP address to your instance. I allocated ${publicIp} for you earlier, and now
I'll associate it to your instance.`
);
associationId = await associateAddress({
  allocationId: ipAllocationId,
  instanceId,
});
console.log(
  "Done. Now you should be able to SSH using the new IP.\n",
  `${displaySSHConnectionInfo({ publicIp, keyPairName })}`
);
await prompter.confirm({ message: confirmMessage });
console.log(
  "I'll restart the server again so you can see the IP address remains the
same."
);
const ipAddressAfterAssociated = await restartInstance(instanceId);
console.log(
  `Done. Here's your SSH info. Notice the IP address hasn't changed.`,
  `
${displaySSHConnectionInfo({
  publicIp: ipAddressAfterAssociated,
  keyPairName,
})}`
);
await prompter.confirm({ message: confirmMessage });
} catch (err) {
  console.error(err);
} finally {
  // Clean up.
  console.log(wrapText("Clean up."));
  console.log("Now I'll clean up all of the stuff I created.");
}
```

```
await prompter.confirm({ message: confirmMessage });
console.log("Cleaning up. Some of these steps can take a bit of time.");
await disassociateAddress(associationId);
await terminateInstance(instanceId);
await releaseAddress(ipAllocationId);
await deleteSecurityGroup(securityGroupId);
deleteTemporaryDirectory();
await deleteKeyPair(keyPairName);
console.log(
  "Done cleaning up. Thanks for staying until the end!",
  "If you have any feedback please use the feedback button in the docs",
  "or create an issue on GitHub.",
);
}
};
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for JavaScript .
 - [AllocateAddress](#)
 - [AssociateAddress](#)
 - [AuthorizeSecurityGroupIngress](#)
 - [CreateKeyPair](#)
 - [CreateSecurityGroup](#)
 - [DeleteKeyPair](#)
 - [DeleteSecurityGroup](#)
 - [DescribeImages](#)
 - [DescribeInstanceTypes](#)
 - [DescribeInstances](#)
 - [DescribeKeyPairs](#)
 - [DescribeSecurityGroups](#)
 - [DisassociateAddress](#)
 - [ReleaseAddress](#)
 - [RunInstances](#)
 - [StartInstances](#)
 - [StopInstances](#)
 - [TerminateInstances](#)

- [UnmonitorInstances](#)

Contoh Elastic Load Balancing menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Elastic Load Balancing.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Elastic Load Balancing

Contoh kode berikut menunjukkan cara memulai menggunakan Elastic Load Balancing.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
```

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new DescribeLoadBalancersCommand({}),
);
const loadBalancersList = LoadBalancers.map(
  (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
).join("\n");
console.log(
  "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
  loadBalancersList,
);
}

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

- Untuk detail API, lihat [DescribeLoadBalancers](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Buat pendengar untuk penyeimbang beban

Contoh kode berikut menunjukkan cara membuat listener yang meneruskan permintaan dari penyeimbang beban ELB ke grup target.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const client = new ElasticLoadBalancingV2Client({});
const { Listeners } = await client.send(
  new CreateListenerCommand({
    LoadBalancerArn: state.loadBalancerArn,
    Protocol: state.targetGroupProtocol,
    Port: state.targetGroupPort,
    DefaultActions: [
      { Type: "forward", TargetGroupArn: state.targetGroupArn },
    ],
  }),
);
```

- Untuk detail API, lihat [CreateListener](#) di Referensi AWS SDK for JavaScript API.

Buat grup target

Contoh kode berikut menunjukkan cara membuat grup target ELB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new CreateTargetGroupCommand({
    Name: NAMES.loadBalancerTargetGroupName,
    Protocol: "HTTP",
    Port: 80,
    HealthCheckPath: "/healthcheck",
    HealthCheckIntervalSeconds: 10,
    HealthCheckTimeoutSeconds: 5,
    HealthyThresholdCount: 2,
    UnhealthyThresholdCount: 2,
    VpcId: state.defaultVpc,
  }),
);
```

- Untuk detail API, lihat [CreateTargetGroup](#) di Referensi AWS SDK for JavaScript API.

Membuat Application Load Balancer

Contoh kode berikut menunjukkan cara membuat ELB Application Load Balancer.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).


```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new CreateLoadBalancerCommand({
    Name: NAMES.loadBalancerName,
    Subnets: state.subnets,
  }),
);
state.loadBalancerDns = LoadBalancers[0].DNSName;
state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
await waitUntilLoadBalancerAvailable(
  { client },
  { Names: [NAMES.loadBalancerName] },
);
```

- Untuk detail API, lihat [CreateLoadBalancer](#) di Referensi AWS SDK for JavaScript API.

Menghapus penyeimbang beban

Contoh kode berikut menunjukkan cara menghapus penyeimbang beban ELB.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).


```
const client = new ElasticLoadBalancingV2Client({});
const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
await client.send(
  new DeleteLoadBalancerCommand({
    LoadBalancerArn: loadBalancer.LoadBalancerArn,
  }),
);
await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
  const lb = await findLoadBalancer(NAMES.loadBalancerName);
  if (lb) {
    throw new Error("Load balancer still exists.");
  }
});
```

- Untuk detail API, lihat [DeleteLoadBalancer](#) di Referensi AWS SDK for JavaScript API.

Menghapus grup target

Contoh kode berikut menunjukkan cara menghapus grup target ELB.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const client = new ElasticLoadBalancingV2Client({});
try {
```

```
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);

await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  client.send(
    new DeleteTargetGroupCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  ),
);
} catch (e) {
  state.deleteLoadBalancerTargetGroupError = e;
}
```

- Untuk detail API, lihat [DeleteTargetGroup](#) di Referensi AWS SDK for JavaScript API.

Jelaskan kelompok sasaran

Contoh kode berikut menunjukkan bagaimana menggambarkan kelompok sasaran tertentu.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- Untuk detail API, lihat [DescribeTargetGroups](#) di Referensi AWS SDK for JavaScript API.

Dapatkan titik akhir dari penyeimbang beban

Contoh kode berikut menunjukkan cara mendapatkan titik akhir dari penyeimbang beban ELB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

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

- Untuk detail API, lihat [DescribeLoadBalancers](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan kesehatan kelompok sasaran

Contoh kode berikut menunjukkan cara mendapatkan kesehatan instance dalam kelompok target ELB.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- Untuk detail API, lihat [DescribeTargetHealth](#) di Referensi AWS SDK for JavaScript API.

Skenario

Membangun dan mengelola layanan yang tangguh

Contoh kode berikut menunjukkan cara membuat layanan web load-balanced yang mengembalikan rekomendasi buku, film, dan lagu. Contoh ini menunjukkan cara layanan tersebut merespons kegagalan, serta cara merestrukturisasi layanan agar lebih tangguh ketika terjadi kegagalan.

- Menggunakan grup Amazon EC2 Auto Scaling untuk membuat instans Amazon Elastic Compute Cloud (Amazon EC2) berdasarkan templat peluncuran dan menyimpan sejumlah instans dalam rentang yang ditentukan.
- Menangani dan mendistribusikan permintaan HTTP dengan Elastic Load Balancing.
- Memantau kondisi instans dalam grup Auto Scaling dan meneruskan permintaan hanya ke instans yang sehat.
- Menjalankan server web Python pada setiap instans EC2 untuk menangani permintaan HTTP. Server web merespons dengan memberikan rekomendasi dan melakukan pemeriksaan kondisi.

- Mensimulasikan layanan yang direkomendasikan dengan tabel Amazon DynamoDB.
- Kontrol respons server web terhadap permintaan dan pemeriksaan kesehatan dengan memperbarui AWS Systems Manager parameter.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menjalankan skenario interaktif di prompt perintah.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 * - deploy
 * - demo
 * - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};
```

```
/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

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

Menyusun langkah-langkah untuk men-deploy semua sumber daya.

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

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
```

```
    DescribeSubnetsCommand,
    DescribeSecurityGroupsCommand,
    AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
    IAMClient,
    CreatePolicyCommand,
    CreateRoleCommand,
    CreateInstanceProfileCommand,
    AddRoleToInstanceProfileCommand,
    AttachRolePolicyCommand,
    waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
    CreateAutoScalingGroupCommand,
    AutoScalingClient,
    AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
    CreateListenerCommand,
    CreateLoadBalancerCommand,
    CreateTargetGroupCommand,
    ElasticLoadBalancingV2Client,
    waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
    ScenarioOutput,
    ScenarioInput,
    ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "../constants.js";
import { initParamsSteps } from "../steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
    new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
    new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
        type: "confirm",
```

```
    }),
    new ScenarioAction(
      "handleConfirmDeployment",
      (c) => c.confirmDeployment === false && process.exit(),
    ),
    new ScenarioOutput(
      "creatingTable",
      MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
    new ScenarioAction("createTable", async () => {
      const client = new DynamoDBClient({});
      await client.send(
        new CreateTableCommand({
          TableName: NAMES.tableName,
          ProvisionedThroughput: {
            ReadCapacityUnits: 5,
            WriteCapacityUnits: 5,
          },
          AttributeDefinitions: [
            {
              AttributeName: "MediaType",
              AttributeType: "S",
            },
            {
              AttributeName: "ItemId",
              AttributeType: "N",
            },
          ],
          KeySchema: [
            {
              AttributeName: "MediaType",
              KeyType: "HASH",
            },
            {
              AttributeName: "ItemId",
              KeyType: "RANGE",
            },
          ],
        })
      );
      await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
    }),
    new ScenarioOutput(
      "createdTable",
```

```

    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(

```

```

    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    ),
  })

```

```
    }},
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
```

```

        InstanceProfileName: NAMES.instanceProfileName,
    })),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
        { client },
        { InstanceProfileName: NAMES.instanceProfileName },
    );
    })),
    new ScenarioOutput("createdInstanceProfile", (state) =>
        MESSAGES.createdInstanceProfile
            .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
            .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
    ),
    new ScenarioOutput(
        "addingRoleToInstanceProfile",
        MESSAGES.addingRoleToInstanceProfile
            .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
    ),
    new ScenarioAction("addRoleToInstanceProfile", () => {
        const client = new IAMClient({});
        return client.send(
            new AddRoleToInstanceProfileCommand({
                RoleName: NAMES.instanceRoleName,
                InstanceProfileName: NAMES.instanceProfileName,
            }),
        );
    }),
    new ScenarioOutput(
        "addedRoleToInstanceProfile",
        MESSAGES.addedRoleToInstanceProfile
            .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
    ),
    ...initParamsSteps,
    new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
    new ScenarioAction("createLaunchTemplate", async () => {
        // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
        const ssmClient = new SSMClient({});
        const { Parameter } = await ssmClient.send(
            new GetParameterCommand({
                Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
            }),
        );
    });
}

```



```
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
      LaunchTemplateData: {
        InstanceType: "t3.micro",
        ImageId: Parameter.Value,
        IamInstanceProfile: { Name: NAMES.instanceProfileName },
        UserData: readFileSync(
          join(RESOURCES_PATH, "server_startup_script.sh"),
        ).toString("base64"),
        KeyName: NAMES.keyPairName,
      },
    }),
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({
        AvailabilityZones: state.availabilityZoneNames,
```

```

        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
        },
        MinSize: 3,
        MaxSize: 3,
    })),
    ),
);
}),
new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
        MESSAGES.createdAutoScalingGroup
            .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
            .replace(
                "${AVAILABILITY_ZONE_NAMES}",
                state.availabilityZoneNames.join(", "),
            ),
    ),
new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
}),
new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
new ScenarioAction("getVpc", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
        new DescribeVpcsCommand({
            Filters: [{ Name: "is-default", Values: ["true"] }],
        }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),

```

```
new ScenarioAction("getSubnets", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
});
```

```

    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
  state.targetGroupProtocol = targetGroup.Protocol;
  state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
  "createdLoadBalancerTargetGroup",
  MESSAGES.createdLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
    })),
  );
  state.loadBalancerDns = LoadBalancers[0].DNSName;
  state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
  await waitUntilLoadBalancerAvailable(
    { client },
    { Names: [NAMES.loadBalancerName] },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
  MESSAGES.createdLoadBalancer
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioOutput(
  "creatingListener",
  MESSAGES.creatingLoadBalancerListener

```

```
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
```

```

    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

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

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**

```

```
    * @param {{ myIpRules: any[] }} state
    */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
        })
      );
    }
  )
);
```

```
        ToPort: 80,
        IpProtocol: "tcp",
    })),
    );
},
),
new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
        return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
        return false;
    }
}),
new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioAction("verifyEndpoint", async (state) => {
    try {
        const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
            axios.get(`http://${state.loadBalancerDns}`),
        );
        state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
        state.verifyEndpointError = e;
    }
}),
new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
        console.error(state.verifyEndpointError);
    } else {
        return MESSAGES.verifiedEndpoint.replace(
            "${ENDPOINT_RESPONSE}",
            state.endpointResponse,
        );
    }
}),
];
```

Menyusun langkah-langkah untuk menjalankan demo.

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



```
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
```

```
import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
});
```

```
);
// snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
    },
    output: getRecommendationResult,
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
```

```
    input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
      type: "confirm",
    }),
    output: getHealthCheckResult,
  },
},
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
```

```
MESSAGES.demoTestBrokenDependency.replace(
  "${TABLE_NAME}",
  state.badTableName,
),
),
...statusSteps,
new ScenarioInput(
  "staticResponseConfirmation",
  MESSAGES.demoStaticResponseConfirmation,
  { type: "confirm" },
),
new ScenarioAction("staticResponse", async (state) => {
  if (!state.staticResponseConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,
        Value: "static",
        Overwrite: true,
        Type: "String",
      })),
    );
  }
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
  "badCredentialsConfirmation",
  MESSAGES.demoBadCredentialsConfirmation,
  { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
  if (!state.badCredentialsConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("fixDynamoDBName", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
```

```

        Overwrite: true,
        Type: "String",
    })),
    );
  })),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
    */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
    });

```

```
// snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

await ec2Client.send(
  new RebootInstancesCommand({
    InstanceIds: [state.targetInstance.InstanceId],
  }),
);

const ssmClient = new SSMClient({});
await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
  const { InstanceInformationList } = await ssmClient.send(
    new DescribeInstanceInformationCommand({}),
  );

  const instance = InstanceInformationList.find(
    (info) => info.InstanceId === state.targetInstance.InstanceId,
  );

  if (!instance) {
    throw new Error("Instance not found.");
  }
});

await ssmClient.send(
  new SendCommandCommand({
    InstanceIds: [state.targetInstance.InstanceId],
    DocumentName: "AWS-RunShellScript",
    Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
  }),
);
},
),
new ScenarioOutput(
  "testBadCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
   */
  (state) =>
    MESSAGES.demoTestBadCredentials.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
),
```

```
),
loadBalancerLoop,
new ScenarioInput(
  "deepHealthCheckConfirmation",
  MESSAGES.demoDeepHealthCheckConfirmation,
  { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
  if (!state.deepHealthCheckConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("deepHealthCheck", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmHealthCheckKey,
      Value: "deep",
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "killInstanceConfirmation",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
   ssm').InstanceInformation }} state
   */
  (state) =>
    MESSAGES.demoKillInstanceConfirmation.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
  { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
  if (!state.killInstanceConfirmation) {
    process.exit();
  }
}),

```



```
new ScenarioAction(
  "killInstance",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
  ssm').InstanceInformation }} state
   */
  async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new TerminateInstanceInAutoScalingGroupCommand({
        InstanceId: state.targetInstance.InstanceId,
        ShouldDecrementDesiredCapacity: false,
      }),
    );
  },
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("failOpen", () => {
  const client = new SSMClient({});
  return client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: `fake-table-${Date.now()}`,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "resetTableConfirmation",
  MESSAGES.demoResetTableConfirmation,
```

```

    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      })
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    })
  );
};

await iamClient.send(
  new CreateRoleCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: { Service: "ec2.amazonaws.com" },
          Action: "sts:AssumeRole",
        }
      ],
    })
  })
);

```

```
    ],
  })),
});
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: Policy.Arn,
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
  }),
);
// snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
// snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
  }),
);

return InstanceProfile;
}
```

Menyusun langkah-langkah untuk menghancurkan semua sumber daya.

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

```
import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
```

```
new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
new ScenarioAction(
  "abort",
  (state) => state.destroy === false && process.exit(),
),
new ScenarioAction("deleteTable", async (c) => {
  try {
    const client = new DynamoDBClient({});
    await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
  } catch (e) {
    c.deleteTableError = e;
  }
}),
new ScenarioOutput("deleteTableResult", (state) => {
  if (state.deleteTableError) {
    console.error(state.deleteTableError);
    return MESSAGES.deleteTableError.replace(
      "${TABLE_NAME}",
      NAMES.tableName,
    );
  } else {
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  }
}),
new ScenarioAction("deleteKeyPair", async (state) => {
  try {
    const client = new EC2Client({});
    await client.send(
      new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
    );
    unlinkSync(`${NAMES.keyPairName}.pem`);
  } catch (e) {
    state.deleteKeyPairError = e;
  }
}),
new ScenarioOutput("deleteKeyPairResult", (state) => {
  if (state.deleteKeyPairError) {
    console.error(state.deleteKeyPairError);
    return MESSAGES.deleteKeyPairError.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  } else {
    return MESSAGES.deletedKeyPair.replace(
```

```
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
    );
}
}),
new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
        const client = new IAMClient({});
        const policy = await findPolicy(NAMES.instancePolicyName);

        if (!policy) {
            state.detachPolicyFromRoleError = new Error(
                `Policy ${NAMES.instancePolicyName} not found.`
            );
        } else {
            await client.send(
                new DetachRolePolicyCommand({
                    RoleName: NAMES.instanceRoleName,
                    PolicyArn: policy.Arn,
                })
            );
        }
    } catch (e) {
        state.detachPolicyFromRoleError = e;
    }
}),
new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
        console.error(state.detachPolicyFromRoleError);
        return MESSAGES.detachPolicyFromRoleError
            .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
        return MESSAGES.detachedPolicyFromRole
            .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
}),
new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
        state.deletePolicyError = new Error(
```

```
        `Policy ${NAMES.instancePolicyName} not found.`,\n    );\n  } else {\n    return client.send(\n      new DeletePolicyCommand({\n        PolicyArn: policy.Arn,\n      }),\n    );\n  }\n}),\nnew ScenarioOutput("deletePolicyResult", (state) => {\n  if (state.deletePolicyError) {\n    console.error(state.deletePolicyError);\n    return MESSAGES.deletePolicyError.replace(\n      "${INSTANCE_POLICY_NAME}",\n      NAMES.instancePolicyName,\n    );\n  } else {\n    return MESSAGES.deletedPolicy.replace(\n      "${INSTANCE_POLICY_NAME}",\n      NAMES.instancePolicyName,\n    );\n  }\n}),\nnew ScenarioAction("removeRoleFromInstanceProfile", async (state) => {\n  try {\n    const client = new IAMClient({});\n    await client.send(\n      new RemoveRoleFromInstanceProfileCommand({\n        RoleName: NAMES.instanceRoleName,\n        InstanceProfileName: NAMES.instanceProfileName,\n      }),\n    );\n  } catch (e) {\n    state.removeRoleFromInstanceProfileError = e;\n  }\n}),\nnew ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {\n  if (state.removeRoleFromInstanceProfile) {\n    console.error(state.removeRoleFromInstanceProfileError);\n    return MESSAGES.removeRoleFromInstanceProfileError\n      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)\n      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);\n  } else {\n
```

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



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

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

```
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  } else {
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      client.send(
        new DeleteTargetGroupCommand({
          TargetGroupArn: TargetGroups[0].TargetGroupArn,
        }),
      ),
    );
  } catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
  }
  // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  } else {
```

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

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

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

```

    );
  }
}),
new ScenarioAction("deleteSsmOnlyRole", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DeleteRoleCommand({
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyRoleError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
  if (state.deleteSsmOnlyRoleError) {
    console.error(state.deleteSsmOnlyRoleError);
    return MESSAGES.deleteSsmOnlyRoleError.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  } else {
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }
}),
];

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

```

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

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



```
}

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

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for JavaScript .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)
 - [DescribeInstanceProfileAssociations](#)
 - [DescribeInstances](#)
 - [DescribeLoadBalancers](#)
 - [DescribeSubnets](#)

- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

EventBridge contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) with EventBridge.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Tambahkan target

Contoh kode berikut menunjukkan cara menambahkan target ke EventBridge acara Amazon.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import {
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
//     totalRetryDelay: 0
//   },
//   FailedEntries: [],
//   FailedEntryCount: 0
// }

return response;
};
```

- Untuk detail API, lihat [PutTargets](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

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

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

```
  }  
});
```

- Untuk detail API, lihat [PutTargets](#) di Referensi AWS SDK for JavaScript API.

Buat aturan

Contoh kode berikut menunjukkan cara membuat EventBridge aturan Amazon.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";  
  
export const putRule = async (  
  ruleName = "some-rule",  
  source = "some-source",  
) => {  
  const client = new EventBridgeClient({});  
  
  const response = await client.send(  
    new PutRuleCommand({  
      Name: ruleName,  
      EventPattern: JSON.stringify({ source: [source] }),  
      State: "ENABLED",  
      EventBusName: "default",  
    })),  
  );  
  
  console.log("PutRule response:");  
  console.log(response);  
  // PutRule response:  
  // {  
  //   '$metadata': {
```

```
//     httpStatusCode: 200,  
//     requestId: 'd7292ced-1544-421b-842f-596326bc7072',  
//     extendedRequestId: undefined,  
//     cfId: undefined,  
//     attempts: 1,  
//     totalRetryDelay: 0  
//   },  
//   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/  
EventBridgeTestRule-1696280037720'  
// }  
return response;  
};
```

- Untuk detail API, lihat [PutRule](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create CloudWatchEvents service object  
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });  
  
var params = {  
  Name: "DEMO_EVENT",  
  RoleArn: "IAM_ROLE_ARN",  
  ScheduleExpression: "rate(5 minutes)",  
  State: "ENABLED",  
};  
  
ebevents.putRule(params, function (err, data) {  
  if (err) {  
    console.log("Error", err);  
  } else {
```

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

- Untuk detail API, lihat [PutRule](#) di Referensi AWS SDK for JavaScript API.

Kirim acara

Contoh kode berikut menunjukkan cara mengirim EventBridge acara Amazon.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
import {
  EventBridgeClient,
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";

export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutEventsCommand({
      Entries: [
        {
          Detail: JSON.stringify({ greeting: "Hello there." }),
          DetailType: detailType,
          Resources: resources,
          Source: source,
        },
      ],
    })
  );
}
```

```
    ],
  })),
);

console.log("PutEvents response:");
console.log(response);
// PutEvents response:
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
//   FailedEntryCount: 0
// }

return response;
};
```

- Untuk detail API, lihat [PutEvents](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

var params = {
```



```
    Entries: [  
      {  
        Detail: '{ "key1": "value1", "key2": "value2" }',  
        DetailType: "appRequestSubmitted",  
        Resources: ["RESOURCE_ARN"],  
        Source: "com.company.app",  
      },  
    ],  
  };  
  
  ebevents.putEvents(params, function (err, data) {  
    if (err) {  
      console.log("Error", err);  
    } else {  
      console.log("Success", data.Entries);  
    }  
  });
```

- Untuk detail API, lihat [PutEvents](#) di Referensi AWS SDK for JavaScript API.

AWS Glue contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) with AWS Glue.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo AWS Glue

Contoh kode berikut menunjukkan cara untuk mulai menggunakan AWS Glue.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});

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

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- Untuk detail API, lihat [ListJobs](#) di Referensi AWS SDK for JavaScript API.

Topik


- [Tindakan](#)
- [Skenario](#)

Tindakan

Buat crawler

Contoh kode berikut menunjukkan cara membuat AWS Glue crawler.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });


  return client.send(command);
};
```

- Untuk detail API, lihat [CreateCrawler](#) di Referensi AWS SDK for JavaScript API.

Buat definisi pekerjaan

Contoh kode berikut menunjukkan cara membuat definisi AWS Glue pekerjaan.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
```

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

const command = new CreateJobCommand({
  Name: name,
  Role: role,
  Command: {
    Name: "glueetl",
    PythonVersion: "3",
    ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
  },
  GlueVersion: "3.0",
});

return client.send(command);
};
```

- Untuk detail API, lihat [CreateJob](#) di Referensi AWS SDK for JavaScript API.

Hapus crawler

Contoh kode berikut menunjukkan cara menghapus AWS Glue crawler.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [DeleteCrawler](#) di Referensi AWS SDK for JavaScript API.

Menghapus database dari Katalog Data

Contoh kode berikut menunjukkan cara menghapus database dari file AWS Glue Data Catalog.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [DeleteDatabase](#) di Referensi AWS SDK for JavaScript API.

Hapus definisi pekerjaan

Contoh kode berikut menunjukkan cara menghapus definisi AWS Glue pekerjaan dan semua proses terkait.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [DeleteJob](#) di Referensi AWS SDK for JavaScript API.

Menghapus tabel dari database

Contoh kode berikut menunjukkan cara menghapus tabel dari AWS Glue Data Catalog database.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for JavaScript API.

Dapatkan crawler

Contoh kode berikut menunjukkan cara mendapatkan AWS Glue crawler.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [GetCrawler](#) di Referensi AWS SDK for JavaScript API.

Dapatkan database dari Katalog Data

Contoh kode berikut menunjukkan bagaimana untuk mendapatkan database dari AWS Glue Data Catalog.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const getDatabase = (name) => {
  const client = new GlueClient({});
```

```
const command = new GetDatabaseCommand({
  Name: name,
});

return client.send(command);
};
```

- Untuk detail API, lihat [GetDatabase](#) di Referensi AWS SDK for JavaScript API.

Dapatkan pekerjaan

Contoh kode berikut menunjukkan cara menjalankan AWS Glue pekerjaan.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });


  return client.send(command);
};
```

- Untuk detail API, lihat [GetJobRun](#) di Referensi AWS SDK for JavaScript API.

Dapatkan database dari Katalog Data

Contoh kode berikut menunjukkan bagaimana untuk mendapatkan daftar database dari. AWS Glue Data Catalog

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});


  return client.send(command);
};
```

- Untuk detail API, lihat [GetDatabases](#) di Referensi AWS SDK for JavaScript API.

Dapatkan pekerjaan dari Katalog Data

Contoh kode berikut menunjukkan cara mendapatkan pekerjaan dari AWS Glue Data Catalog.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [GetJob](#) di Referensi AWS SDK for JavaScript API.

Menjalankan pekerjaan

Contoh kode berikut menunjukkan cara menjalankan AWS Glue pekerjaan.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [GetJobRuns](#) di Referensi AWS SDK for JavaScript API.

Dapatkan tabel dari database

Contoh kode berikut menunjukkan bagaimana untuk mendapatkan tabel dari database di AWS Glue Data Catalog.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [GetTables](#) di Referensi AWS SDK for JavaScript API.

Daftar definisi pekerjaan

Contoh kode berikut menunjukkan bagaimana daftar definisi AWS Glue pekerjaan.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- Untuk detail API, lihat [ListJobs](#) di Referensi AWS SDK for JavaScript API.

Mulai crawler

Contoh kode berikut menunjukkan cara memulai AWS Glue crawler.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });


  return client.send(command);
};
```

- Untuk detail API, lihat [StartCrawler](#) di Referensi AWS SDK for JavaScript API.

Memulai eksekusi tugas

Contoh kode berikut menunjukkan cara memulai AWS Glue pekerjaan.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
    },
  });

  return client.send(command);
};
```

```
    "--input_table": tableName,  
    "--output_bucket_url": `s3://${bucketName}/`,  
  },  
});  
  
return client.send(command);  
};
```

- Untuk detail API, lihat [StartJobRun](#) di Referensi AWS SDK for JavaScript API.

Skenario

Memulai crawler dan lowongan

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat crawler yang merayapi bucket Amazon S3 publik dan membuat database metadata berformat CSV.
- Daftar informasi tentang database dan tabel di Anda AWS Glue Data Catalog.
- Buat pekerjaan untuk mengekstrak data CSV dari bucket S3, mengubah data, dan memuat output berformat JSON ke bucket S3 lain.
- Buat daftar informasi tentang menjalankan pekerjaan, melihat data yang diubah, dan membersihkan sumber daya.

Untuk informasi selengkapnya, lihat [Tutorial: Memulai AWS Glue Studio](#).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Membuat dan menjalankan crawler yang merayapi bucket Amazon Simple Storage Service (Amazon S3) publik dan menghasilkan database metadata yang menjelaskan data berformat CSV yang ditemukannya.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
  } catch {
    return false;
  }
}
```

```
};

const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH
    );

    log("Crawler created successfully.", { type: "success" });
  }

  return { ...context };
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }

  log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
  await wait(waitTimeInSeconds);
  return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
  ({ startCrawler, getCrawler }) =>
```

```
async (context) => {
  log("Starting crawler.");
  await startCrawler(process.env.CRAWLER_NAME);
  log("Crawler started.", { type: "success" });

  log("Waiting for crawler to finish running. This can take a while.");
  await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
  log("Crawler ready.", { type: "success" });

  return { ...context };
};
```

Daftar informasi tentang database dan tabel di Anda AWS Glue Data Catalog.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};

const makeGetDatabaseStep =
  ({ getDatabase }) =>
  async (context) => {
    const {
      Database: { Name },
    } = await getDatabase(process.env.DATABASE_NAME);
    log(`Database: ${Name}`);
    return { ...context };
  };
```



```
};

const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
```

Buat dan jalankan job yang mengekstrak data CSV dari bucket Amazon S3 sumber, mengubahnya dengan menghapus dan mengganti nama bidang, dan memuat output berformat JSON ke bucket Amazon S3 lainnya.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  },
```

```
});

return client.send(command);
};

const makeCreateJobStep =
({ createJob }) =>
async (context) => {
  log("Creating Job.");
  await createJob(
    process.env.JOB_NAME,
    process.env.ROLE_NAME,
    process.env.BUCKET_NAME,
    process.env.PYTHON_SCRIPT_KEY,
  );
  log("Job created.", { type: "success" });

  return { ...context };
};

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput> } getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }

  switch (JobRun.JobRunState) {
    case "FAILED":
    case "TIMEOUT":
    case "STOPPED":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "RUNNING":
      break;
    case "SUCCEEDED":
  }
}
```

```
        return;
    default:
        throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
    }

    log(
        `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
    );
    await wait(waitTimeInSeconds);
    return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
    const { shouldOpen } = await context.prompter.prompt({
        name: "shouldOpen",
        type: "confirm",
        message: "Open the output bucket in your browser?",
    });

    if (shouldOpen) {
        return open(
            `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
            view the output.`
        );
    }
};

const makeStartJobRunStep =
    ({ startJobRun, getJobRun }) =>
    async (context) => {
        log("Starting job.");
        const { JobRunId } = await startJobRun(
            process.env.JOB_NAME,
            process.env.DATABASE_NAME,
            process.env.TABLE_NAME,
            process.env.BUCKET_NAME,
        );
        log("Job started.", { type: "success" });

        log("Waiting for job to finish running. This can take a while.");
        await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    }
};
```

```
log("Job run succeeded.", { type: "success" });

await promptToOpen(context);

return { ...context };
};
```

Buat daftar informasi tentang pekerjaan berjalan dan lihat beberapa data yang diubah.

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};

const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
      });
    }
  }
};
```

```
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
    });

    logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
}

return { ...context };
};
```

Hapus semua sumber daya yang dibuat oleh demo.

```
const deleteJob = (jobName) => {
    const client = new GlueClient({});

    const command = new DeleteJobCommand({
        JobName: jobName,
    });

    return client.send(command);
};

const deleteTable = (databaseName, tableName) => {
    const client = new GlueClient({});

    const command = new DeleteTableCommand({
        DatabaseName: databaseName,
        Name: tableName,
    });

    return client.send(command);
};

const deleteDatabase = (databaseName) => {
    const client = new GlueClient({});

    const command = new DeleteDatabaseCommand({
        Name: databaseName,
    });

    return client.send(command);
};
```

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};

const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error))
    );
    log("Jobs deleted.", { type: "success" });
  }
};

const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
```

```
        deleteTable(databaseName, tableName).catch(console.error)
    )
);

const makeCleanUpTablesStep =
({ getTables, deleteTable }) =>
async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
        () => ({ TableList: null })
    );

    if (TableList && TableList.length > 0) {
        const { tableNames } = await context.prompter.prompt({
            name: "tableNames",
            type: "checkbox",
            message: "Let's clean up tables. Select tables to delete.",
            choices: TableList.map((t) => t.Name),
        });

        if (tableNames.length === 0) {
            log("No tables selected.");
        } else {
            log("Deleting tables.");
            await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
            log("Tables deleted.", { type: "success" });
        }
    }

    return { ...context };
};

const deleteDatabases = (deleteDatabase, databaseNames) =>
Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error))
);

const makeCleanUpDatabasesStep =
({ getDatabases, deleteDatabase }) =>
async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
        const { dbNames } = await context.prompter.prompt({
            name: "dbNames",
```

```
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
    });

    if (dbNames.length === 0) {
        log("No databases selected.");
    } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
    }
}

return { ...context };
};

const cleanUpCrawlerStep = async (context) => {
    log(`Deleting crawler.`);

    try {
        await deleteCrawler(process.env.CRAWLER_NAME);
        log("Crawler deleted.", { type: "success" });
    } catch (err) {
        if (err.name === "EntityNotFoundException") {
            log(`Crawler is already deleted.`);
        } else {
            throw err;
        }
    }

    return { ...context };
};
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .

- [CreateCrawler](#)
- [CreateJob](#)
- [DeleteCrawler](#)
- [DeleteDatabase](#)
- [DeleteJob](#)

- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)
- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

HealthImaging contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) with HealthImaging.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo HealthImaging

Contoh kode berikut menunjukkan cara untuk mulai menggunakan HealthImaging.

SDK untuk JavaScript (v3)

```
import {
```

```
ListDatastoresCommand,  
MedicalImagingClient,  
} from "@aws-sdk/client-medical-imaging";  
  
// When no region or credentials are provided, the SDK will use the  
// region and credentials from the local AWS config.  
const client = new MedicalImagingClient({});  
  
export const helloMedicalImaging = async () => {  
  const command = new ListDatastoresCommand({});  
  
  const { datastoreSummaries } = await client.send(command);  
  console.log("Datastores: ");  
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));  
  return datastoreSummaries;  
};
```

- Untuk detail API, lihat [ListDatastores](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Penambahan sebuah tag ke sebuah sumber daya

Contoh kode berikut menunjukkan cara menambahkan tag ke HealthImaging sumber daya.

SDK untuk JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";  
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
```

```
/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *
 *     - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  // }

  return response;
};
```

- Untuk detail API, lihat [TagResource](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Salin satu set gambar

Contoh kode berikut menunjukkan cara menyalin set HealthImaging gambar.

SDK untuk JavaScript (v3)

Fungsi utilitas untuk menyalin set gambar.

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 * set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 * image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```

//      requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
//      extendedRequestId: undefined,
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    datastoreId: 'xxxxxxxxxxxxxxxx',
//    destinationImageSetProperties: {
//      createdAt: 2023-09-27T19:46:21.824Z,
//      imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxxxx',
//      imageSetId: 'xxxxxxxxxxxxxxxx',
//      imageSetState: 'LOCKED',
//      imageSetWorkflowStatus: 'COPYING',
//      latestVersionId: '1',
//      updatedAt: 2023-09-27T19:46:21.824Z
//    },
//    sourceImageSetProperties: {
//      createdAt: 2023-09-22T14:49:26.427Z,
//      imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxxxx',
//      imageSetId: 'xxxxxxxxxxxxxxxx',
//      imageSetState: 'LOCKED',
//      imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
//      latestVersionId: '4',
//      updatedAt: 2023-09-27T19:46:21.824Z
//    }
//  }
return response;
};

```

Salin set gambar tanpa tujuan.

```

try {
  await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.error(err);
}

```

Salin set gambar dengan tujuan.

```
try {
  await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "4",
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.error(err);
}
```

- Untuk detail API, lihat [CopyImageSet](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat penyimpanan data

Contoh kode berikut menunjukkan cara membuat penyimpanan HealthImaging data.

SDK untuk JavaScript (v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName })
  );
  console.log(response);
}
```

```
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   datastoreStatus: 'CREATING'
// }
return response;
};
```

- Untuk detail API, lihat [CreateDatastore](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus penyimpanan data

Contoh kode berikut menunjukkan cara menghapus penyimpanan HealthImaging data.

SDK untuk JavaScript (v3)

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId })
  );
  console.log(response);
  // {
  //   '$metadata': {
```

```

//      httpStatusCode: 200,
//      requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
//      extendedRequestId: undefined,
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//    datastoreStatus: 'DELETING'
// }

return response;
};

```

- Untuk detail API, lihat [DeleteDatastore](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus satu set gambar

Contoh kode berikut menunjukkan cara menghapus set HealthImaging gambar.

SDK untuk JavaScript (v3)

```

import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,

```



```

        imageSetId: imageSetId,
    })
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   datastoreId: 'xxxxxxxxxxxxxxxxxxxx',
//   imageSetId: 'xxxxxxxxxxxxxxxxxxxx',
//   imageSetState: 'LOCKED',
//   imageSetWorkflowStatus: 'DELETING'
// }
return response;
};

```

- Untuk detail API, lihat [DeleteImageSet](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan bingkai gambar

Contoh kode berikut menunjukkan cara mendapatkan bingkai gambar.

SDK untuk JavaScript (v3)

```

import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 * image frame.
 * @param {string} datastoreID - The data store's ID.

```

```
* @param {string} imageSetID - The image set's ID.
* @param {string} imageFrameID - The image frame's ID.
*/
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreId = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   contentType: 'application/octet-stream',
  //   imageFrameBlob: <ref *1> IncomingMessage {}
  // }
  return response;
};
```

- Untuk detail API, lihat [GetImageFrame](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan properti penyimpanan data

Contoh kode berikut menunjukkan cara mendapatkan properti penyimpanan HealthImaging data.

SDK untuk JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreProperties: {
  //     createdAt: 2023-08-04T18:50:36.239Z,
  //     datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreName: 'my_datastore',
  //     datastoreStatus: 'ACTIVE',
  //     updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response["datastoreProperties"];
};
```

- Untuk detail API, lihat [GetDatastore](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan properti set gambar

Contoh kode berikut menunjukkan bagaimana untuk mendapatkan properti HealthImaging image set.

SDK untuk JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  // }
```



```

    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     statusCode: 200,
  //     requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   jobProperties: {
  //     dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     endedAt: 2023-09-19T17:29:21.753Z,
  //     inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobName: 'job_1',
  //     jobStatus: 'COMPLETED',
  //     outputS3Uri: 's3://health-imaging-dest/
  output_ct/'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'/',
  //     submittedAt: 2023-09-19T17:27:25.143Z
  //   }
  // }

  return response;
};

```

- Untuk detail API, lihat [GetDicom ImportJob](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan metadata untuk kumpulan gambar

Contoh kode berikut menunjukkan cara mendapatkan metadata untuk kumpulan HealthImaging gambar.

SDK untuk JavaScript (v3)

Fungsi utilitas untuk mendapatkan metadata set gambar.

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   contentType: 'application/json',
  //   contentEncoding: 'gzip',

```

```
//   imageSetMetadataBlob: <ref *1> IncomingMessage {}  
// }  
  
return response;  
};
```

Dapatkan metadata set gambar tanpa versi.

```
try {  
  await getImageSetMetadata(  
    "metadata.json.gzip",  
    "12345678901234567890123456789012",  
    "12345678901234567890123456789012"  
  );  
} catch (err) {  
  console.log("Error", err);  
}
```

Dapatkan metadata set gambar dengan versi.

```
try {  
  await getImageSetMetadata(  
    "metadata2.json.gzip",  
    "12345678901234567890123456789012",  
    "12345678901234567890123456789012",  
    "1"  
  );  
} catch (err) {  
  console.log("Error", err);  
}
```

- Untuk detail API, lihat [GetImageSetMetadata](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Impor data massal ke penyimpanan data

Contoh kode berikut menunjukkan cara mengimpor data massal ke penyimpanan HealthImaging data.

SDK untuk JavaScript (v3)

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/"
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
```

```
// },
//   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   jobStatus: 'SUBMITTED',
//   submittedAt: 2023-09-22T14:48:45.767Z
// }
return response;
};
```

- Untuk detail API, lihat [StartDicom ImportJob](#) di AWS SDK for JavaScript Referensi API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Daftar penyimpanan data

Contoh kode berikut menunjukkan cara daftar penyimpanan HealthImaging data.

SDK untuk JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    larger than `pageSize`.
  }
}
```


SDK untuk JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    // larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   imageSetPropertiesList: [
  //     {
  //       ImageSetWorkflowStatus: 'CREATED',
  //       createdAt: 2023-09-22T14:49:26.427Z,
```

```

//          imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//          imageSetState: 'ACTIVE',
//          versionId: '1'
//      }]
// }
return imageSetPropertiesList;
};

```

- Untuk detail API, lihat [ListImageSetVersions](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Daftar pekerjaan impor untuk penyimpanan data

Contoh kode berikut menunjukkan cara daftar pekerjaan impor untuk penyimpanan HealthImaging data.

SDK untuk JavaScript (v3)

```

import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];

```

```
for await (const page of paginator) {
  // Each page contains a list of `jobSummaries`. The list is truncated if is
  // larger than `pageSize`.
  jobSummaries.push(...page["jobSummaries"]);
  console.log(page);
}
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   jobSummaries: [
//     {
//       dataAccessRoleArn: 'arn:aws:iam:xxxxxxxxxxxx:role/dicom_import',
//       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       endedAt: 2023-09-22T14:49:51.351Z,
//       jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       jobName: 'test-1',
//       jobStatus: 'COMPLETED',
//       submittedAt: 2023-09-22T14:48:45.767Z
//     }
//   ]
// }

return jobSummaries;
};
```

- Untuk detail API, lihat [ListDicom ImportJobs](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Membuat daftar tanda untuk sumber daya

Contoh kode berikut menunjukkan bagaimana untuk daftar tag untuk HealthImaging sumber daya.

SDK untuk JavaScript (v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- Untuk detail API, lihat [ListTagsForResource](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menghapus tag dari sumber daya

Contoh kode berikut menunjukkan cara menghapus tag dari HealthImaging sumber daya.

SDK untuk JavaScript (v3)

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
  xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- Untuk detail API, lihat [UntagResource](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Cari set gambar

Contoh kode berikut menunjukkan cara mencari set HealthImaging gambar.

SDK untuk JavaScript (v3)

Fungsi utilitas untuk mencari set gambar.

```
import { paginateSearchImageSets } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters - The
  search criteria filters.
 */
export const searchImageSets = async (
  datastoreId = "xxxxxxxx",
  filters = []
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {
    datastoreId: datastoreId,
    searchCriteria: {
      filters,
    },
  };

  const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

  const imageSetsMetadataSummaries = [];
  for await (const page of paginator) {
```

```
// Each page contains a list of `jobSummaries`. The list is truncated if is
larger than `pageSize`.
imageSetsMetadataSummaries.push(...page["imageSetsMetadataSummaries"]);
console.log(page);
}
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   imageSetsMetadataSummaries: [
//     {
//       DICOMTags: [Object],
//       createdAt: "2023-09-19T16:59:40.551Z",
//       imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',
//       updatedAt: "2023-09-19T16:59:40.551Z",
//       version: 1
//     }
//   ]
// }

return imageSetsMetadataSummaries;
};
```

Kasus penggunaan #1: operator EQUAL.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [{ DICOMPatientId: "9227465" }],
      operator: "EQUAL",
    },
  ],
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}
```

```
}
```

Kasus penggunaan #2: ANTARA operator menggunakan DICOM StudyDate dan StudyTime DICOM.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "19900101",
            DICOMStudyTime: "000000",
          },
        },
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "20230901",
            DICOMStudyTime: "000000",
          },
        },
      ],
      operator: "BETWEEN",
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}
```

Kasus penggunaan #3: ANTARA operator menggunakan createDat. Studi waktu sebelumnya bertahan.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
```

```

    {
      values: [
        { createdAt: new Date("1985-04-12T23:20:50.52Z") },
        { createdAt: new Date("2023-09-12T23:20:50.52Z") },
      ],
      operator: "BETWEEN",
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}

```

- Untuk detail API, lihat [SearchImageSets](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Perbarui metadata set gambar

Contoh kode berikut menunjukkan cara memperbarui HealthImaging metadata set gambar.

SDK untuk JavaScript (v3)

```

import {UpdateImageSetMetadataCommand} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}') => {
  const response = await medicalImagingClient.send(

```

```

    new UpdateImageSetMetadataCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
      latestVersionId: latestVersionId,
      updateImageSetMetadataUpdates: updateMetadata
    })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   createdAt: 2023-09-22T14:49:26.427Z,
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   imageSetState: 'LOCKED',
  //   imageSetWorkflowStatus: 'UPDATING',
  //   latestVersionId: '4',
  //   updatedAt: 2023-09-27T19:41:43.494Z
  // }
  return response;
};

```

Mengkodekan metadata.

```

const updatableAttributes =
JSON.stringify({
  "SchemaVersion": 1.1,
  "Patient": {
    "DICOM": {
      "PatientName": "Garcia^Gloria"
    }
  }
})

const updateMetadata = {
  "DICOMUpdates": {

```

```
        "updatableAttributes":  
            new TextEncoder().encode(updatableAttributes)  
        }  
    };  
  
    await updateImageSetMetadata("12345678901234567890123456789012",  
    "12345678901234567890123456789012",  
    "1", updateMetadata);
```

- Untuk detail API, lihat [UpdateImageSetMetadata](#) di Referensi AWS SDK for JavaScript API.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Skenario

Menandai penyimpanan data

Contoh kode berikut menunjukkan cara menandai penyimpanan HealthImaging data.

SDK untuk JavaScript (v3)

Untuk menandai penyimpanan data.

```
try {  
    const datastoreArn =  
        "arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012";  
    const tags = {  
        Deployment: "Development",  
    };  
    await tagResource(datastoreArn, tags);  
} catch (e) {  
    console.log(e);  
}
```

Fungsi utilitas untuk menandai sumber daya.

```

import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 * - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  // }

  return response;
};

```

Untuk daftar tag untuk penyimpanan data.

```

try {
  const datastoreArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
  const { tags } = await listTagsForResource(datastoreArn);
  console.log(tags);
} catch (e) {
  console.log(e);
}

```

```
}
```

Fungsi utilitas untuk daftar tag sumber daya.

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
  // }

  return response;
};
```

Untuk menghapus tag penyimpanan data.

```
try {
  const datastoreArn =
    "arn:aws:medical-imaging:us-east-1:123456789012:datastore/12345678901234567890123456789012";
  const keys = ["Deployment"];
  await untagResource(datastoreArn, keys);
}
```



```
    } catch (e) {  
      console.log(e);  
    }  
  }
```

Fungsi utilitas untuk membuka tag sumber daya.

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";  
import { medicalImagingClient } from "../libs/medicalImagingClient.js";  
  
/**  
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store  
 or image set.  
 * @param {string[]} tagKeys - The keys of the tags to remove.  
 */  
export const untagResource = async (  
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/  
xxx",  
  tagKeys = []  
) => {  
  const response = await medicalImagingClient.send(  
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })  
  );  
  console.log(response);  
  // {  
  //   '$metadata': {  
  //     httpStatusCode: 204,  
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',  
  //     extendedRequestId: undefined,  
  //     cfId: undefined,  
  //     attempts: 1,  
  //     totalRetryDelay: 0  
  //   }  
  // }  
  
  return response;  
};
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [ListTagsForResource](#)
 - [TagResource](#)

- [UntagResource](#)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menandai set gambar

Contoh kode berikut menunjukkan cara menandai set HealthImaging gambar.

SDK untuk JavaScript (v3)

Untuk menandai set gambar.

```
try {
  const imagesetArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
  const tags = {
    Deployment: "Development",
  };
  await tagResource(imagesetArn, tags);
} catch (e) {
  console.log(e);
}
```

Fungsi utilitas untuk menandai sumber daya.

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 * - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
```

```

    resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
    tags = {}
) => {
    const response = await medicalImagingClient.send(
        new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
    );
    console.log(response);
    // {
    //     '$metadata': {
    //         httpStatusCode: 204,
    //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     }
    // }
    // }

    return response;
};

```

Untuk mencantumkan tag untuk kumpulan gambar.

```

try {
    const imagesetArn =
        "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(imagesetArn);
    console.log(tags);
} catch (e) {
    console.log(e);
}

```

Fungsi utilitas untuk daftar tag sumber daya.

```

import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**

```

```

* @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
or image set.
*/
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
  // }

  return response;
};

```

Untuk menghapus tag set gambar.

```

try {
  const imagesetArn =
    "arn:aws:medical-imaging:us-east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
  const keys = ["Deployment"];
  await untagResource(imagesetArn, keys);
} catch (e) {
  console.log(e);
}

```

Fungsi utilitas untuk membuka tag sumber daya.

```

import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";

```

```
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh IAM menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan IAM.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo IAM

Contoh kode berikut menunjukkan bagaimana memulai menggunakan IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
   * async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
```

```
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
  for await (const page of paginator) {
    if (page.Policies) {
      page.Policies.forEach((p) => {
        console.log(`${p.PolicyName}`);
        policyCount++;
      });
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- Untuk detail API, lihat [ListPolicies](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Lampirkan kebijakan ke peran

Contoh kode berikut menunjukkan cara melampirkan kebijakan IAM ke peran.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Lampirkan kebijakan.

```
import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [AttachRolePolicy](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};
```



```
iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        console.log(
          "AmazonDynamoDBFullAccess is already attached to this role."
        );
        process.exit();
      }
    });
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.attachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [AttachRolePolicy](#) di Referensi AWS SDK for JavaScript API.

Lampirkan kebijakan inline ke peran

Contoh kode berikut menunjukkan cara melampirkan kebijakan inline ke peran IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::some-test-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
```

```
    PolicyName: policyName,  
    PolicyDocument: policyDocument,  
  });  
  
  const response = await client.send(command);  
  console.log(response);  
  return response;  
};
```

- Untuk detail API, lihat [PutRolePolicy](#) di Referensi AWS SDK for JavaScript API.

Buat penyedia SAFL

Contoh kode berikut menunjukkan cara membuat penyedia SAFL AWS Identity and Access Management (IAM).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";  
import { readFileSync } from "fs";  
import * as path from "path";  
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";  
  
const client = new IAMClient({});  
  
/**  
 * This sample document was generated using Auth0.  
 * For more information on generating this document,  
 * see https://docs.aws.amazon.com/IAM/latest/UserGuide/  
id_roles_providers_create_saml.html#samlstep1.  
 */  
const sampleMetadataDocument = readFileSync(  
  path.join(  
    dirnameFromMetaUrl(import.meta.url),
```

```
    "../../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });

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

- Untuk detail API, lihat [CreateSamlProvider di Referensi](#) API.AWS SDK for JavaScript

Membuat grup

Contoh kode berikut menunjukkan cara membuat grup IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
```

```
* @param {string} groupName
*/
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

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

- Untuk detail API, lihat [CreateGroup](#) di Referensi AWS SDK for JavaScript API.

Buat kebijakan

Contoh kode berikut menunjukkan cara membuat kebijakan IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat kebijakan.

```
import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyName
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
```

```
        Action: "*",
        Resource: "*",
    },
],
}),
PolicyName: policyName,
});

return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreatePolicy](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
```

```
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
    ],
    Resource: "RESOURCE_ARN",
  },
],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreatePolicy](#) di Referensi AWS SDK for JavaScript API.

Membuat peran

Contoh kode berikut menunjukkan cara membuat peran IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat peran.

```
import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
  const command = new CreateRoleCommand({
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {
            Service: "lambda.amazonaws.com",
          },
          Action: "sts:AssumeRole",
        },
      ],
    }),
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [CreateRole](#) di Referensi AWS SDK for JavaScript API.

Buat peran tertaut layanan

Contoh kode berikut menunjukkan cara membuat peran terkait layanan IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat peran terkait layanan.

```
import { CreateServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-
    // that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/
    // latest/gr/aws-service-information.html.
    AWSServiceName: serviceName,
  });

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

- Untuk detail API, lihat [CreateServiceLinkedRole](#) di Referensi AWS SDK for JavaScript API.

Buat pengguna

Contoh kode berikut menunjukkan cara membuat pengguna IAM.

Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat pengguna.

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const createUser = (name) => {
  const command = new CreateUserCommand({ UserName: name });
  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateUser](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

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

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateUser](#) di Referensi AWS SDK for JavaScript API.

Buat kunci akses

Contoh kode berikut menunjukkan cara membuat kunci akses IAM.

Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat kunci akses.

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateAccessKey](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });


iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateAccessKey](#) di Referensi AWS SDK for JavaScript API.

Buat alias untuk akun

Contoh kode berikut menunjukkan cara membuat alias untuk akun IAM.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat alias akun.

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias - A unique name for the account alias.
 * @returns
 */
export const createAccountAlias = (alias) => {
  const command = new CreateAccountAliasCommand({
    AccountAlias: alias,
```

```
});  
  
    return client.send(command);  
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateAccountAlias](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).


```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create the IAM service object  
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });  
  
iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {  
    if (err) {  
        console.log("Error", err);  
    } else {  
        console.log("Success", data);  
    }  
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateAccountAlias](#) di Referensi AWS SDK for JavaScript API.

Buat profil instance

Contoh kode berikut menunjukkan cara membuat profil instance IAM.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).


```
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```

- Untuk detail API, lihat [CreateInstanceProfile](#) di Referensi AWS SDK for JavaScript API.

Hapus penyedia SAFL

Contoh kode berikut menunjukkan cara menghapus penyedia SAFL AWS Identity and Access Management (IAM).

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
```

```
* @param {string} providerArn
* @returns
*/
export const deleteSAMLProvider = async (providerArn) => {
  const command = new DeleteSAMLProviderCommand({
    SAMLProviderArn: providerArn,
  });

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

- Untuk detail API, lihat [DeleteSamlProvider di Referensi](#) API.AWS SDK for JavaScript

Menghapus grup

Contoh kode berikut menunjukkan cara menghapus grup IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
    GroupName: groupName,
  });

  const response = await client.send(command);
```



```
console.log(response);
return response;
};
```

- Untuk detail API, lihat [DeleteGroup](#) di Referensi AWS SDK for JavaScript API.

Menghapus kebijakan

Contoh kode berikut menunjukkan cara menghapus kebijakan IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus kebijakan.

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});


/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- Untuk detail API, lihat [DeletePolicy](#) di Referensi AWS SDK for JavaScript API.

Hapus peran

Contoh kode berikut menunjukkan cara menghapus peran IAM.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus peran.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});


/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Untuk detail API, lihat [DeleteRole](#) di Referensi AWS SDK for JavaScript API.

Menghapus kebijakan peran

Contoh kode berikut menunjukkan cara menghapus kebijakan peran IAM.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";
```

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

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
  });
  return client.send(command);
};
```

- Untuk detail API, lihat [DeleteRolePolicy](#) di Referensi AWS SDK for JavaScript API.

Hapus sertifikat server

Contoh kode berikut menunjukkan cara menghapus sertifikat server IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus sertifikat server.

```
import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
```

```
const command = new DeleteServerCertificateCommand({
  ServerCertificateName: certName,
});

return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteServerCertificate](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteServerCertificate](#) di Referensi AWS SDK for JavaScript API.

Hapus peran tertaut layanan

Contoh kode berikut menunjukkan cara menghapus peran terkait layanan IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Untuk detail API, lihat [DeleteServiceLinkedRole](#) di Referensi AWS SDK for JavaScript API.

Hapus pengguna

Contoh kode berikut menunjukkan cara menghapus pengguna IAM.

Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus pengguna.

```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
  const command = new DeleteUserCommand({ UserName: name });
  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteUser](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

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

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteUser](#) di Referensi AWS SDK for JavaScript API.


Hapus kunci akses

Contoh kode berikut menunjukkan cara menghapus kunci akses IAM.

Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus tombol akses.

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";


const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const deleteAccessKey = (userName, accessKeyId) => {
  const command = new DeleteAccessKeyCommand({
    AccessKeyId: accessKeyId,
    UserName: userName,
  });

  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteAccessKey](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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



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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteAccessKey](#) di Referensi AWS SDK for JavaScript API.

Hapus alias akun

Contoh kode berikut menunjukkan cara menghapus alias akun IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus alias akun.

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

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

```
/**
 *
 * @param {string} alias
 */
export const deleteAccountAlias = (alias) => {
  const command = new DeleteAccountAliasCommand({ AccountAlias: alias });

  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteAccountAlias](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteAccountAlias](#) di Referensi AWS SDK for JavaScript API.

Hapus profil instance

Contoh kode berikut menunjukkan cara menghapus profil instans IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const client = new IAMClient({});
await client.send(
  new DeleteInstanceProfileCommand({
    InstanceProfileName: NAMES.instanceProfileName,
  }),
);
```

- Untuk detail API, lihat [DeleteInstanceProfile](#) di Referensi AWS SDK for JavaScript API.

Lepaskan kebijakan dari peran

Contoh kode berikut menunjukkan cara melepaskan kebijakan IAM dari peran.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
*
* @param {string} policyArn
* @param {string} roleName
*/
export const detachRolePolicy = (policyArn, roleName) => {
  const command = new DetachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DetachRolePolicy](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
```

```
myRolePolicies.forEach(function (val, index, array) {
  if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.detachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to detach policy from role", err);
      } else {
        console.log("Policy detached from role successfully");
        process.exit();
      }
    });
  }
});
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DetachRolePolicy](#) di Referensi AWS SDK for JavaScript API.

Dapatkan kebijakan

Contoh kode berikut menunjukkan cara mendapatkan kebijakan IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan kebijakan.

```
import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
*
* @param {string} policyArn
*/
export const getPolicy = (policyArn) => {
  const command = new GetPolicyCommand({
    PolicyArn: policyArn,
  });

  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetPolicy](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};


iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetPolicy](#) di Referensi AWS SDK for JavaScript API.

Dapatkan peran

Contoh kode berikut menunjukkan cara mendapatkan peran IAM.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan perannya.

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const getRole = (roleName) => {
  const command = new GetRoleCommand({
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [GetRole](#) di Referensi AWS SDK for JavaScript API.

Dapatkan sertifikat server

Contoh kode berikut menunjukkan cara mendapatkan sertifikat server IAM.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan sertifikat server.

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";


const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetServerCertificate](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).


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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetServerCertificate](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan status penghapusan peran terkait layanan

Contoh kode berikut menunjukkan cara mendapatkan status penghapusan peran terkait layanan AWS Identity and Access Management (IAM).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
```

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

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [GetServiceLinkedRoleDeletionStatus](#) di Referensi AWS SDK for JavaScript API.

Dapatkan data tentang penggunaan terakhir kunci akses

Contoh kode berikut menunjukkan cara mendapatkan data tentang penggunaan terakhir kunci akses IAM.

Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan kunci akses.

```
import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} accessKeyId
 */
export const getAccessKeyLastUsed = async (accessKeyId) => {
  const command = new GetAccessKeyLastUsedCommand({
    AccessKeyId: accessKeyId,
  });

  const response = await client.send(command);

  if (response.AccessKeyLastUsed?.LastUsedDate) {
    console.log(`
    ${accessKeyId} was last used by ${response.UserName} via
    the ${response.AccessKeyLastUsed.ServiceName} service on
    ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
    `);
  }

  return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetAccessKeyLastUsed](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getAccessKeyLastUsed(
  { AccessKeyId: "ACCESS_KEY_ID" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
    }
  }
);
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetAccessKeyLastUsed](#) di Referensi AWS SDK for JavaScript API.

Dapatkan kebijakan kata sandi akun

Contoh kode berikut menunjukkan cara mendapatkan kebijakan kata sandi akun IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan kebijakan kata sandi akun.

```
import {
  GetAccountPasswordPolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const getAccountPasswordPolicy = async () => {
  const command = new GetAccountPasswordPolicyCommand({});
```

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

- Untuk detail API, lihat [GetAccountPasswordPolicy](#) di Referensi AWS SDK for JavaScript API.

Daftar penyedia SALL

Contoh kode berikut menunjukkan bagaimana untuk daftar penyedia SALL untuk IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Daftar penyedia SAFL.

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listSamlProviders = async () => {
  const command = new ListSAMLProvidersCommand({});

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

- Untuk detail API, lihat [ListSamlProviders](#) di AWS SDK for JavaScript Referensi API.

Daftar kunci akses pengguna

Contoh kode berikut menunjukkan bagaimana untuk daftar kunci akses IAM pengguna.

⚠ Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

ℹ Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar kunci akses.

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
  const command = new ListAccessKeysCommand({
    MaxItems: 5,
    UserName: userName,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.AccessKeyMetadata?.length) {
```

```
for (const key of response.AccessKeyMetadata) {
  yield key;
}

if (response.IsTruncated) {
  response = await client.send(
    new ListAccessKeysCommand({
      Marker: response.Marker,
    }),
  );
} else {
  break;
}
}
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListAccessKeys](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};

iam.listAccessKeys(params, function (err, data) {
```

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListAccessKeys](#) di Referensi AWS SDK for JavaScript API.

Daftar alias akun

Contoh kode berikut menunjukkan bagaimana untuk daftar alias akun IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar alias akun.

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.
 */
export async function* listAccountAliases() {
  const command = new ListAccountAliasesCommand({ MaxItems: 5 });

  let response = await client.send(command);

  while (response.AccountAliases?.length) {
```



```
for (const alias of response.AccountAliases) {
  yield alias;
}

if (response.IsTruncated) {
  response = await client.send(
    new ListAccountAliasesCommand({
      Marker: response.Marker,
      MaxItems: 5,
    }),
  );
} else {
  break;
}
}
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListAccountAliases](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

```
  }  
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListAccountAliases](#) di Referensi AWS SDK for JavaScript API.

Daftar grup

Contoh kode berikut menunjukkan bagaimana daftar grup IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Daftar grup.

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 * A generator function that handles paginated results.  
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/  
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify  
 this.  
 */  
export async function* listGroups() {  
  const command = new ListGroupsCommand({  
    MaxItems: 10,  
  });  
  
  let response = await client.send(command);  
  
  while (response.Groups?.length) {  
    for (const group of response.Groups) {  
      yield group;  
    }  
  }  
}
```

```
    if (response.IsTruncated) {
      response = await client.send(
        new ListGroupsCommand({
          Marker: response.Marker,
          MaxItems: 10,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Untuk detail API, lihat [ListGroups](#) di Referensi AWS SDK for JavaScript API.

Daftar kebijakan inline untuk peran

Contoh kode berikut menunjukkan cara membuat daftar kebijakan inline untuk peran IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar kebijakan.

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.
 *
 * @param {string} roleName
```

```
*/
export async function* listRolePolicies(roleName) {
  const command = new ListRolePoliciesCommand({
    RoleName: roleName,
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.PolicyNames?.length) {
    for (const policyName of response.PolicyNames) {
      yield policyName;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolePoliciesCommand({
          RoleName: roleName,
          MaxItems: 10,
          Marker: response.Marker,
        })),
    );
    } else {
      break;
    }
  }
}
```

- Untuk detail API, lihat [ListRolePolicies](#) di Referensi AWS SDK for JavaScript API.

Daftar kebijakan

Contoh kode berikut menunjukkan cara membuat daftar kebijakan IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar kebijakan.

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
 * AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 * this.
 */
export async function* listPolicies() {
  const command = new ListPoliciesCommand({
    MaxItems: 10,
    OnlyAttached: false,
    // List only the customer managed policies in your Amazon Web Services account.
    Scope: "Local",
  });

  let response = await client.send(command);

  while (response.Policies?.length) {
    for (const policy of response.Policies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListPoliciesCommand({
          Marker: response.Marker,
          MaxItems: 10,
          OnlyAttached: false,
          Scope: "Local",
        })),
    );
  } else {
    break;
  }
}
```

- Untuk detail API, lihat [ListPolicies](#) di Referensi AWS SDK for JavaScript API.

Daftar kebijakan yang dilampirkan pada peran

Contoh kode berikut menunjukkan cara membuat daftar kebijakan yang dilampirkan ke peran IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar kebijakan yang dilampirkan pada peran.

```
import {
  ListAttachedRolePoliciesCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
 * AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 * this.
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
  const command = new ListAttachedRolePoliciesCommand({
    RoleName: roleName,
  });

  let response = await client.send(command);

  while (response.AttachedPolicies?.length) {
    for (const policy of response.AttachedPolicies) {
      yield policy;
    }
  }
}
```

```
    if (response.IsTruncated) {
      response = await client.send(
        new ListAttachedRolePoliciesCommand({
          RoleName: roleName,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Untuk detail API, lihat [ListAttachedRolePolicies](#) di Referensi AWS SDK for JavaScript API.

Daftar peran

Contoh kode berikut menunjukkan bagaimana untuk daftar peran IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar peran.

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.
 */
export async function* listRoles() {
```

```
const command = new ListRolesCommand({
  MaxItems: 10,
});

/**
 * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}
 */
let response = await client.send(command);

while (response?.Roles?.length) {
  for (const role of response.Roles) {
    yield role;
  }

  if (response.IsTruncated) {
    response = await client.send(
      new ListRolesCommand({
        Marker: response.Marker,
      }),
    );
  } else {
    break;
  }
}
}
```

- Untuk detail API, lihat [ListRoles](#) di Referensi AWS SDK for JavaScript API.

Daftar sertifikat server

Contoh kode berikut menunjukkan bagaimana untuk daftar sertifikat server IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar sertifikat.


```
import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
this.
 *
 */
export async function* listServerCertificates() {
  const command = new ListServerCertificatesCommand({});
  let response = await client.send(command);

  while (response.ServerCertificateMetadataList?.length) {
    for await (const cert of response.ServerCertificateMetadataList) {
      yield cert;
    }

    if (response.IsTruncated) {
      response = await client.send(new ListServerCertificatesCommand({}));
    } else {
      break;
    }
  }
}
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListServerCertificates](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListServerCertificates](#) di Referensi AWS SDK for JavaScript API.

Daftar pengguna

Contoh kode berikut menunjukkan bagaimana untuk daftar pengguna IAM.

Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Daftar pengguna.

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";
```

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

export const listUsers = async () => {
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);
  response.Users?.forEach(({ UserName, CreateDate }) => {
    console.log(`${UserName} created on: ${CreateDate}`);
  });
  return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListUsers](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
  }
});
```

```
users.forEach(function (user) {
  console.log("User " + user.UserName + " created", user.CreateDate);
});
}
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListUsers](#) di Referensi AWS SDK for JavaScript API.

Memperbarui sertifikat server

Contoh kode berikut menunjukkan cara memperbarui sertifikat server IAM.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Perbarui sertifikat server.

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentName
 * @param {string} newName
 */
export const updateServerCertificate = (currentName, newName) => {
  const command = new UpdateServerCertificateCommand({
    ServerCertificateName: currentName,
    NewServerCertificateName: newName,
  });

  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [UpdateServerCertificate](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh selengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [UpdateServerCertificate](#) di Referensi AWS SDK for JavaScript API.

Memperbarui pengguna

Contoh kode berikut menunjukkan cara memperbarui pengguna IAM.

⚠ Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

📘 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Perbarui pengguna.

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentUserName
 * @param {string} newUserName
 */
export const updateUser = (currentUserName, newUserName) => {
  const command = new UpdateUserCommand({
    UserName: currentUserName,
    NewUserName: newUserName,
  });

  return client.send(command);
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [UpdateUser](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [UpdateUser](#) di Referensi AWS SDK for JavaScript API.

Perbarui kunci akses

Contoh kode berikut menunjukkan cara memperbarui kunci akses IAM.

⚠ Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

SDK untuk JavaScript (v3)

i Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Perbarui kunci akses.

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const updateAccessKey = (userName, accessKeyId) => {
  const command = new UpdateAccessKeyCommand({
    AccessKeyId: accessKeyId,
    Status: StatusType.Inactive,
    UserName: userName,
  });

  return client.send(command);
};
```


- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [UpdateAccessKey](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [UpdateAccessKey](#) di Referensi AWS SDK for JavaScript API.

Unggah sertifikat server

Contoh kode berikut menunjukkan cara meng-upload sertifikat server AWS Identity and Access Management (IAM).

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }
  }

  throw err;
};
```

```
/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [UploadServerCertificate](#) di Referensi AWS SDK for JavaScript API.

Skenario

Membangun dan mengelola layanan yang tangguh

Contoh kode berikut menunjukkan cara membuat layanan web load-balanced yang mengembalikan rekomendasi buku, film, dan lagu. Contoh ini menunjukkan cara layanan tersebut merespons kegagalan, serta cara merestrukturisasi layanan agar lebih tangguh ketika terjadi kegagalan.

- Menggunakan grup Amazon EC2 Auto Scaling untuk membuat instans Amazon Elastic Compute Cloud (Amazon EC2) berdasarkan templat peluncuran dan menyimpan sejumlah instans dalam rentang yang ditentukan.
- Menangani dan mendistribusikan permintaan HTTP dengan Elastic Load Balancing.
- Memantau kondisi instans dalam grup Auto Scaling dan meneruskan permintaan hanya ke instans yang sehat.
- Menjalankan server web Python pada setiap instans EC2 untuk menangani permintaan HTTP. Server web merespons dengan memberikan rekomendasi dan melakukan pemeriksaan kondisi.
- Menyimulasikan layanan yang direkomendasikan dengan tabel Amazon DynamoDB.
- Kontrol respons server web terhadap permintaan dan pemeriksaan kesehatan dengan memperbarui AWS Systems Manager parameter.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menjalankan skenario interaktif di prompt perintah.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 * - deploy
 * - demo
 * - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

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

Menyusun langkah-langkah untuk men-deploy semua sumber daya.

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

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
```

```
    CreatePolicyCommand,
    CreateRoleCommand,
    CreateInstanceProfileCommand,
    AddRoleToInstanceProfileCommand,
    AttachRolePolicyCommand,
    waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
    CreateAutoScalingGroupCommand,
    AutoScalingClient,
    AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
    CreateListenerCommand,
    CreateLoadBalancerCommand,
    CreateTargetGroupCommand,
    ElasticLoadBalancingV2Client,
    waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
    ScenarioOutput,
    ScenarioInput,
    ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "../constants.js";
import { initParamsSteps } from "../steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
    new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
    new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
        type: "confirm",
    }),
    new ScenarioAction(
        "handleConfirmDeployment",
        (c) => c.confirmDeployment === false && process.exit(),
    ),
    new ScenarioOutput(
```

```
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),

```

```
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
  const { KeyMaterial } = await client.send(
    new CreateKeyPairCommand({
      KeyName: NAMES.keyPairName,
    }),
  );

  writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
```



```
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  );
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
```

```
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
```

```
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
new ScenarioAction("createLaunchTemplate", async () => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(
    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
```

```
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
      UserData: readFileSync(
        join(RESOURCES_PATH, "server_startup_script.sh"),
      ).toString("base64"),
      KeyName: NAMES.keyPairName,
    },
  })),
  // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
);
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({
        AvailabilityZones: state.availabilityZoneNames,
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
          LaunchTemplateName: NAMES.launchTemplateName,
          Version: "$Default",
        },
      },
      {
        MinSize: 3,
      }
    )
  );
});
```

```

        MaxSize: 3,
      })),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
   */
  (state) =>
    MESSAGES.createdAutoScalingGroup
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
      .replace(
        "${AVAILABILITY_ZONE_NAMES}",
        state.availabilityZoneNames.join(", "),
      ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }]},
      ),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [

```

```
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
    ],
    }),
);
// snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
        MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
    ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
        new CreateTargetGroupCommand({
            Name: NAMES.loadBalancerTargetGroupName,
            Protocol: "HTTP",
            Port: 80,
            HealthCheckPath: "/healthcheck",
            HealthCheckIntervalSeconds: 10,
            HealthCheckTimeoutSeconds: 5,
            HealthyThresholdCount: 2,
            UnhealthyThresholdCount: 2,
            VpcId: state.defaultVpc,
        }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
```

```
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
```

```
const { Listeners } = await client.send(
  new CreateListenerCommand({
    LoadBalancerArn: state.loadBalancerArn,
    Protocol: state.targetGroupProtocol,
    Port: state.targetGroupPort,
    DefaultActions: [
      { Type: "forward", TargetGroupArn: state.targetGroupArn },
    ],
  }),
);
// snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
const listener = Listeners[0];
state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
  MESSAGES.createdLoadBalancerListener.replace(
    "${LB_LISTENER_ARN}",
    state.loadBalancerListenerArn,
  ),
),
new ScenarioOutput(
  "attachingLoadBalancerTargetGroup",
  MESSAGES.attachingLoadBalancerTargetGroup
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
    .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
),
new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
}),
new ScenarioOutput(
  "attachedLoadBalancerTargetGroup",
  MESSAGES.attachedLoadBalancerTargetGroup,
),
new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
new ScenarioAction(
  "verifyInboundPort",
```



```

/**
 *
 * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
 */
async (state) => {
  const client = new EC2Client({});
  const { SecurityGroups } = await client.send(
    new DescribeSecurityGroupsCommand({
      Filters: [{ Name: "group-name", Values: ["default"] }],
    }),
  );
  if (!SecurityGroups) {
    state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
  }
  state.defaultSecurityGroup = SecurityGroups[0];

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

  state.myIpRules = myIpRules;
},
),
new ScenarioOutput(
  "verifiedInboundPort",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return MESSAGES.foundIpRules.replace(
        "${IP_RULES}",

```

```

        JSON.stringify(state.myIpRules, null, 2),
    );
    } else {
        return MESSAGES.noIpRules;
    }
},
),
new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
        if (state.myIpRules.length > 0) {
            return false;
        } else {
            return MESSAGES.noIpRules;
        }
    },
    { type: "confirm" },
),
new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
     */
    async (state) => {
        if (!state.shouldAddInboundRule) {
            return;
        }

        const client = new EC2Client({});
        await client.send(
            new AuthorizeSecurityGroupIngressCommand({
                GroupId: state.defaultSecurityGroup.GroupId,
                CidrIp: `${state.myIp}/32`,
                FromPort: 80,
                ToPort: 80,
                IpProtocol: "tcp",
            }),
        );
    },
),
),

```

```

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

```

Menyusun langkah-langkah untuk menjalankan demo.

```

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

import axios from "axios";

import {

```

```
    DescribeTargetGroupsCommand,  
    DescribeTargetHealthCommand,  
    ElasticLoadBalancingV2Client,  
} from "@aws-sdk/client-elastic-load-balancing-v2";  
import {  
    DescribeInstanceInformationCommand,  
    PutParameterCommand,  
    SSMClient,  
    SendCommandCommand,  
} from "@aws-sdk/client-ssm";  
import {  
    IAMClient,  
    CreatePolicyCommand,  
    CreateRoleCommand,  
    AttachRolePolicyCommand,  
    CreateInstanceProfileCommand,  
    AddRoleToInstanceProfileCommand,  
    waitUntilInstanceProfileExists,  
} from "@aws-sdk/client-iam";  
import {  
    AutoScalingClient,  
    DescribeAutoScalingGroupsCommand,  
    TerminateInstanceInAutoScalingGroupCommand,  
} from "@aws-sdk/client-auto-scaling";  
import {  
    DescribeIamInstanceProfileAssociationsCommand,  
    EC2Client,  
    RebootInstancesCommand,  
    ReplaceIamInstanceProfileAssociationCommand,  
} from "@aws-sdk/client-ec2";  
  
import {  
    ScenarioAction,  
    ScenarioInput,  
    ScenarioOutput,  
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";  
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";  
  
import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";  
import { findLoadBalancer } from "./shared.js";  
  
const getRecommendation = new ScenarioAction(  
    "getRecommendation",  
    async (state) => {
```

```
const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
if (loadBalancer) {
  state.loadBalancerDnsName = loadBalancer.DNSName;
  try {
    state.recommendation = (
      await axios.get(`http://${state.loadBalancerDnsName}`)
    ).data;
  } catch (e) {
    state.recommendation = e instanceof Error ? e.message : e;
  }
} else {
  throw new Error(MESSAGES.demoFindLoadBalancerError);
}
},
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
```

```
"getHealthCheckResult",
/**
 * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
 */
(state) => {
  const status = state.targetHealthDescriptions
    .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
    .join("\n");
  return `Health check:\n${status}`;
},
{ preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);
```

```
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
```

```
new ScenarioInput(
  "staticResponseConfirmation",
  MESSAGES.demoStaticResponseConfirmation,
  { type: "confirm" },
),
new ScenarioAction("staticResponse", async (state) => {
  if (!state.staticResponseConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,
        Value: "static",
        Overwrite: true,
        Type: "String",
      }),
    );
  }
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
  "badCredentialsConfirmation",
  MESSAGES.demoBadCredentialsConfirmation,
  { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
  if (!state.badCredentialsConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("fixDynamoDBName", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioAction(
```



```

    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
    */
    async (state) => {
        await createSsmOnlyInstanceProfile();
        const autoScalingClient = new AutoScalingClient({});
        const { AutoScalingGroups } = await autoScalingClient.send(
            new DescribeAutoScalingGroupsCommand({
                AutoScalingGroupNames: [NAMES.autoScalingGroupName],
            }),
        );
        state.targetInstance = AutoScalingGroups[0].Instances[0];
        // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
        const ec2Client = new EC2Client({});
        const { IamInstanceProfileAssociations } = await ec2Client.send(
            new DescribeIamInstanceProfileAssociationsCommand({
                Filters: [
                    { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
                ],
            }),
        );
        // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
        state.instanceProfileAssociationId =
            IamInstanceProfileAssociations[0].AssociationId;
        // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            ec2Client.send(
                new ReplaceIamInstanceProfileAssociationCommand({
                    AssociationId: state.instanceProfileAssociationId,
                    IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
                }),
            ),
        );
        // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

        await ec2Client.send(
            new RebootInstancesCommand({
                InstanceIds: [state.targetInstance.InstanceId],
            }),
        );
    }
}

```

```
    }),
  );

  const ssmClient = new SSMClient({});
  await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
    const { InstanceInformationList } = await ssmClient.send(
      new DescribeInstanceInformationCommand({}),
    );

    const instance = InstanceInformationList.find(
      (info) => info.InstanceId === state.targetInstance.InstanceId,
    );

    if (!instance) {
      throw new Error("Instance not found.");
    }
  });

  await ssmClient.send(
    new SendCommandCommand({
      InstanceIds: [state.targetInstance.InstanceId],
      DocumentName: "AWS-RunShellScript",
      Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
    }),
  );
},
),
new ScenarioOutput(
  "testBadCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
  */
  (state) =>
    MESSAGES.demoTestBadCredentials.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
),
loadBalancerLoop,
new ScenarioInput(
  "deepHealthCheckConfirmation",
  MESSAGES.demoDeepHealthCheckConfirmation,
  { type: "confirm" },

```

```
),
new ScenarioAction("deepHealthCheckExit", (state) => {
  if (!state.deepHealthCheckConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("deepHealthCheck", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmHealthCheckKey,
      Value: "deep",
      Overwrite: true,
      Type: "String",
    })
  ),
);
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "killInstanceConfirmation",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
   ssm').InstanceInformation }} state
   */
  (state) =>
    MESSAGES.demoKillInstanceConfirmation.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
  { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
  if (!state.killInstanceConfirmation) {
    process.exit();
  }
}),
new ScenarioAction(
  "killInstance",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
   ssm').InstanceInformation }} state
   */
```

```
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  })
}
```

```
    }),
    new ScenarioAction("resetTable", async () => {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: NAMES.tableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }),
    new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
    healthCheckLoop,
    loadBalancerLoop,
  ];

  async function createSsmOnlyInstanceProfile() {
    const iamClient = new IAMClient({});
    const { Policy } = await iamClient.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.ssmOnlyPolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "ssm_only_policy.json"),
        ),
      }),
    );
    await iamClient.send(
      new CreateRoleCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Principal: { Service: "ec2.amazonaws.com" },
              Action: "sts:AssumeRole",
            },
          ],
        }),
      }),
    );
    await iamClient.send(
      new AttachRolePolicyCommand({
```

```
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: Policy.Arn,
    })),
);
await iamClient.send(
    new AttachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    })),
);
// snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    })),
);
await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
// snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
await iamClient.send(
    new AddRoleToInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
    })),
);

return InstanceProfile;
}
```

Menyusun langkah-langkah untuk menghancurkan semua sumber daya.

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

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
    EC2Client,
    DeleteKeyPairCommand,
    DeleteLaunchTemplateCommand,
```

```
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
```

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



```
try {
  const client = new IAMClient({});
  const policy = await findPolicy(NAMES.instancePolicyName);

  if (!policy) {
    state.detachPolicyFromRoleError = new Error(
      `Policy ${NAMES.instancePolicyName} not found.`
    );
  } else {
    await client.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: policy.Arn,
      }),
    );
  }
} catch (e) {
  state.detachPolicyFromRoleError = e;
}
}),
new ScenarioOutput("detachedPolicyFromRole", (state) => {
  if (state.detachPolicyFromRoleError) {
    console.error(state.detachPolicyFromRoleError);
    return MESSAGES.detachPolicyFromRoleError
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const policy = await findPolicy(NAMES.instancePolicyName);

  if (!policy) {
    state.deletePolicyError = new Error(
      `Policy ${NAMES.instancePolicyName} not found.`
    );
  } else {
    return client.send(
      new DeletePolicyCommand({
        PolicyArn: policy.Arn,
```

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

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

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

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

```
    );
  } else {
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      client.send(
        new DeleteTargetGroupCommand({
          TargetGroupArn: TargetGroups[0].TargetGroupArn,
        }),
      ),
    );
  } catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
  }
  // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  } else {
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }
}),
```

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

```

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



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

```
    await iamClient.send(
      new DeleteRoleCommand({
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyRoleError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
  if (state.deleteSsmOnlyRoleError) {
    console.error(state.deleteSsmOnlyRoleError);
    return MESSAGES.deleteSsmOnlyRoleError.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  } else {
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }
}),
];

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

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
try {
  await client.send(
    new DeleteAutoScalingGroupCommand({
      AutoScalingGroupName: groupName,
    }),
  );
} catch (err) {
  if (!(err instanceof Error)) {
    throw err;
  } else {
    console.log(err.name);
    throw err;
  }
}

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

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
```


```
const group = page.AutoScalingGroups.find(
  (g) => g.AutoScalingGroupName === groupName,
);
if (group) {
  return group;
}
}
throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for JavaScript .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)
 - [DescribeIamInstanceProfileAssociations](#)
 - [DescribeInstances](#)
 - [DescribeLoadBalancers](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGroups](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)

- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Buat pengguna dan ambil peran


Contoh kode berikut menunjukkan cara membuat pengguna dan mengambil peran.

 Warning

Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

- Buat pengguna tanpa izin.
- Buat peran yang memberikan izin untuk mencantumkan bucket Amazon S3 untuk akun tersebut.
- Tambahkan kebijakan agar pengguna dapat mengambil peran tersebut.
- Asumsikan peran dan daftar bucket S3 menggunakan kredensial sementara, lalu bersihkan sumber daya.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat pengguna IAM dan peran yang memberikan izin untuk mencantumkan bucket Amazon S3. Pengguna hanya memiliki hak untuk mengambil peran. Setelah mengasumsikan peran, gunakan kredensial sementara untuk membuat daftar bucket untuk akun.

```
import {
  CreateUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
```

```
    CreateRoleCommand,
    AttachRolePolicyCommand,
    DeleteAccessKeyCommand,
    DeleteUserCommand,
    DeleteRoleCommand,
    DeletePolicyCommand,
    DetachRolePolicyCommand,
    IAMClient,
  } from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

export const main = async () => {
  // Create a user. The user has no permissions by default.
  const { User } = await iamClient.send(
    new CreateUserCommand({ UserName: userName }),
  );

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
  // STS).
  // It's not best practice to use access keys. For more information, see https://aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }
}
```

```
const {
  AccessKey: { AccessKeyId, SecretAccessKey },
} = createAccessKeyResponse;

let s3Client = new S3Client({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
// thrown while the user and access keys are still stabilizing.
await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
  try {
    return await listBuckets(s3Client);
  } catch (err) {
    if (err instanceof Error && err.name === "InvalidAccessKeyId") {
      throw err;
    }
  }
});

// Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
// is thrown while the user and access keys are still stabilizing.
const { Role } = await retry(
  {
    intervalInMs: 2000,
    maxRetries: 60,
  },
  () =>
    iamClient.send(
      new CreateRoleCommand({
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                // Allow the previously created user to assume this role.
                AWS: User.Arn,
              },
            },
          ],
        }),
      })
    )
  )
);
```

```
        Action: "sts:AssumeRole",
      },
    ],
  )),
  RoleName: roleName,
}),
),
);

if (!Role) {
  throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["s3:ListAllMyBuckets"],
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  }),
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

// Assume the role.
const stsClient = new STSClient({
```



```
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
```

```
        RoleName: Role.RoleName,
    })),
);

await iamClient.send(
    new DeletePolicyCommand({
        PolicyArn: listBucketPolicy.Arn,
    }),
);

await iamClient.send(
    new DeleteRoleCommand({
        RoleName: Role.RoleName,
    }),
);

await iamClient.send(
    new DeleteAccessKeyCommand({
        UserName: userName,
        AccessKeyId,
    }),
);

await iamClient.send(
    new DeleteUserCommand({
        UserName: userName,
    }),
);
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
    const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

    if (!Buckets) {
        throw new Error("Buckets not listed");
    }

    console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Contoh Lambda menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Lambda.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Lambda

Contoh kode berikut menunjukkan cara memulai menggunakan Lambda.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Untuk detail API, lihat [ListFunctions](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Buat fungsi

Contoh kode berikut menunjukkan cara membuat fungsi Lambda.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [CreateFunction](#) di Referensi AWS SDK for JavaScript API.

Hapus fungsi

Contoh kode berikut menunjukkan cara menghapus fungsi Lambda.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Untuk detail API, lihat [DeleteFunction](#) di Referensi AWS SDK for JavaScript API.

Dapatkan fungsi

Contoh kode berikut menunjukkan cara mendapatkan fungsi Lambda.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Untuk detail API, lihat [GetFunction](#) di Referensi AWS SDK for JavaScript API.

Memanggil fungsi

Contoh kode berikut menunjukkan cara memanggil fungsi Lambda.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- Untuk detail API, lihat [Memanggil di Referensi AWS SDK for JavaScript API](#).

Daftar fungsi

Contoh kode berikut menunjukkan bagaimana untuk daftar fungsi Lambda.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});
```

```
    return client.send(command);
  };
```

- Untuk detail API, lihat [ListFunctions](#) di Referensi AWS SDK for JavaScript API.

Perbarui kode fungsi

Contoh kode berikut menunjukkan cara memperbarui kode fungsi Lambda.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Untuk detail API, lihat [UpdateFunctionCode](#) di Referensi AWS SDK for JavaScript API.

Perbarui konfigurasi fungsi

Contoh kode berikut menunjukkan cara memperbarui konfigurasi fungsi Lambda.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}./functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

- Untuk detail API, lihat [UpdateFunctionConfiguration](#) di Referensi AWS SDK for JavaScript API.

Skenario

Memulai dengan fungsi

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat peran IAM dan fungsi Lambda, lalu unggah kode handler.
- Panggil fungsi dengan satu parameter dan dapatkan hasil.
- Perbarui kode fungsi dan konfigurasikan dengan variabel lingkungan.
- Panggil fungsi dengan parameter baru dan dapatkan hasil. Tampilkan log eksekusi yang dikembalikan.
- Buat daftar fungsi untuk akun Anda, lalu bersihkan sumber daya.

Untuk informasi selengkapnya, lihat [Membuat fungsi Lambda dengan konsol](#).

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat peran AWS Identity and Access Management (IAM) yang memberikan izin Lambda untuk menulis ke log.

```
log(`Creating role (${NAME_ROLE_LAMBDA})...`);
const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

Buat fungsi Lambda dan unggah kode handler.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
```

```

    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

```

Panggil fungsi dengan satu parameter dan dapatkan hasil.

```

const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};

```

Perbarui kode fungsi dan konfigurasi lingkungan Lambda dengan variabel lingkungan.

```

const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

```

```
};

const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

Buat daftar fungsi untuk akun Anda.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Hapus peran IAM dan fungsi Lambda.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
```

```
const command = new DeleteFunctionCommand({ FunctionName: funcName });
return client.send(command);
};
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Memohon](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

Amazon Personalisasi contoh menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon Personalize.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Buat pekerjaan antarmuka batch

Contoh kode berikut menunjukkan cara membuat pekerjaan antarmuka batch Amazon Personalize.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchInferenceJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch inference job's parameters.

export const createBatchInferenceJobParam = {
  jobName: 'JOB_NAME',
  jobInput: { /* required */
    s3DataSource: { /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */
    }
  },
  jobOutput: { /* required */
    s3DataDestination: { /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional integer*/
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
      CreateBatchInferenceJobCommand(createBatchInferenceJobParam));
    console.log("Success", response);
  }
}
```

```
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateBatchInferenceJob](#) di Referensi AWS SDK for JavaScript API.

Buat pekerjaan segmen batch

Contoh kode berikut menunjukkan cara membuat pekerjaan segmen batch Amazon Personalize.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: 'NAME',
  jobInput: { /* required */
    s3DataSource: { /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */
    }
  },
},
```

```
jobOutput: {          /* required */
  s3DataDestination: { /* required */
    path: 'OUTPUT_PATH', /* required */
    // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */
  }
},
roleArn: 'ROLE_ARN', /* required */
solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
numResults: 20 /* optional */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateBatchSegmentJobCommand(createBatchSegmentJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateBatchSegmentJob](#) di Referensi AWS SDK for JavaScript API.

Buat kampanye

Contoh kode berikut menunjukkan cara membuat kampanye Amazon Personalize.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.

import { CreateCampaignCommand } from
"@aws-sdk/client-personalize";
```



```
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the campaign's parameters.
export const createCampaignParam = {
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  name: 'NAME', /* required */
  minProvisionedTPS: 1 /* optional integer */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateCampaignCommand(createCampaignParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateCampaign](#) di Referensi AWS SDK for JavaScript API.

Buat kumpulan data

Contoh kode berikut menunjukkan cara membuat dataset Amazon Personalize.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetCommand } from
"@aws-sdk/client-personalize";
```

```
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset's parameters.
export const createDatasetParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  datasetType: 'DATASET_TYPE', /* required */
  name: 'NAME', /* required */
  schemaArn: 'SCHEMA_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateDatasetCommand(createDatasetParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateDataset](#) di Referensi AWS SDK for JavaScript API.

Buat pekerjaan ekspor dataset

Contoh kode berikut menunjukkan cara membuat pekerjaan ekspor dataset Amazon Personalize.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
```

```
import { CreateDatasetExportJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the export job parameters.
export const datasetExportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  jobOutput: {
    s3DataDestination: {
      path: 'S3_DESTINATION_PATH' /* required */
      //kmsKeyArn: 'ARN' /* include if your bucket uses AWS KMS for encryption
    }
  },
  jobName: 'NAME', /* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateDatasetExportJobCommand(datasetExportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateDatasetExportJob](#) di Referensi AWS SDK for JavaScript API.

Membuat grup dataset

Contoh kode berikut menunjukkan cara membuat grup dataset Amazon Personalize.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.

import { CreateDatasetGroupCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset group parameters.
export const createDatasetGroupParam = {
  name: 'NAME' /* required */
}

export const run = async (createDatasetGroupParam) => {
  try {
    const response = await personalizeClient.send(new
    CreateDatasetGroupCommand(createDatasetGroupParam));
    console.log("Success", response);
    return "Run successfully"; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run(createDatasetGroupParam);
```

Buat grup dataset domain.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetGroupCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
```

```
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the domain dataset group parameters.
export const domainDatasetGroupParams = {
  name: 'NAME', /* required */
  domain: 'DOMAIN' /* required for a domain dsG, specify ECOMMERCE or
VIDEO_ON_DEMAND */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
CreateDatasetGroupCommand(domainDatasetGroupParams));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateDatasetGroup](#) di Referensi AWS SDK for JavaScript API.

Buat pekerjaan impor dataset

Contoh kode berikut menunjukkan cara membuat pekerjaan impor dataset Amazon Personalize.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import {CreateDatasetImportJobCommand } from
"@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
```

```
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset import job parameters.
export const datasetImportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  dataSource: { /* required */
    dataLocation: 'S3_PATH'
  },
  jobName: 'NAME', /* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateDatasetImportJobCommand(datasetImportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateDatasetImportJob](#) di Referensi AWS SDK for JavaScript API.

Buat skema domain

Contoh kode berikut menunjukkan cara membuat skema domain Amazon Personalize.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
```

```
"@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema, /* required */
  domain: 'DOMAIN' /* required for a domain dataset group, specify ECOMMERCE or
  VIDEO_ON_DEMAND */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateSchemaCommand(createDomainSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateSchema](#) di Referensi AWS SDK for JavaScript API.

Buat filter

Contoh kode berikut menunjukkan cara membuat filter Amazon Personalize.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateFilterCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the filter's parameters.
export const createFilterParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
  filterExpression: 'FILTER_EXPRESSION' /*required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateFilterCommand(createFilterParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateFilter](#) di Referensi AWS SDK for JavaScript API.

Buat pemberi rekomendasi

Contoh kode berikut menunjukkan cara membuat pemberi rekomendasi Amazon Personalize.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateRecommenderCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the recommender's parameters.
export const createRecommenderParam = {
  name: 'NAME', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  datasetGroupArn: 'DATASET_GROUP_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateRecommenderCommand(createRecommenderParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateRecommender](#) di Referensi AWS SDK for JavaScript API.

Buat skema

Contoh kode berikut menunjukkan cara membuat skema Amazon Personalize.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // For unit tests.
}

// Set the schema parameters.
export const createSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema /* required */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateSchemaCommand(createSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

- Untuk detail API, lihat [CreateSchema](#) di Referensi AWS SDK for JavaScript API.

Buat solusi

Contoh kode berikut menunjukkan cara membuat solusi Amazon Personalize.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution parameters.
export const createSolutionParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  name: 'NAME' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
  CreateSolutionCommand(createSolutionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```


```
run();
```

- Untuk detail API, lihat [CreateSolution](#) di Referensi AWS SDK for JavaScript API.

Buat versi solusi

Contoh kode berikut menunjukkan cara membuat solusi Amazon Personalize.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionVersionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution version parameters.
export const solutionVersionParam = {
  solutionArn: 'SOLUTION_ARN' /* required */
}


export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateSolutionVersionCommand(solutionVersionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateSolutionVersion](#) di Referensi AWS SDK for JavaScript API.

Buat pelacak acara

Contoh kode berikut menunjukkan cara membuat pelacak peristiwa Amazon Personalize.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { CreateEventTrackerCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the event tracker's parameters.
export const createEventTrackerParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateEventTrackerCommand(createEventTrackerParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [CreateEventTracker](#) di Referensi AWS SDK for JavaScript API.

Contoh Personalisasi Acara Amazon menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon Personalize Events.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Impor item ke dalam kumpulan data

Contoh kode berikut menunjukkan cara mengimpor item secara bertahap ke dalam kumpulan data Amazon Personalize Events.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { PutItemsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});
```

```
// Set the put items parameters. For string properties and values, use the \
character to escape quotes.
var putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /* required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
"PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [PutItems](#) di Referensi AWS SDK for JavaScript API.

Impor data peristiwa interaksi waktu nyata

Contoh kode berikut menunjukkan cara mengimpor data peristiwa interaksi real-time ke Amazon Personalize Events.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
Replace it with your sentAt timestamp in UNIX format.

// Set put events parameters.
var putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [PutEvents](#) di Referensi AWS SDK for JavaScript API.

Impor pengguna secara bertahap

Contoh kode berikut menunjukkan cara mengimpor pengguna secara bertahap ke Amazon Personalize Events Events.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { PutUsersCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put users parameters. For string properties and values, use the \
character to escape quotes.
var putUsersParam = {
  datasetArn: "DATASET_ARN",
  users: [
    {
      userId: "USER_ID",
      properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [PutUsers](#) di Referensi AWS SDK for JavaScript API.

Amazon Personalisasi contoh Runtime menggunakan SDK for (v3) JavaScript

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon Personalize Runtime.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Dapatkan rekomendasi (grup dataset kustom)

Contoh kode berikut menunjukkan cara mendapatkan rekomendasi peringkat Amazon Personalize Runtime Runtime.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
```

```
import { GetPersonalizedRankingCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the ranking request parameters.
export const getPersonalizedRankingParam = {
  campaignArn: "CAMPAIGN_ARN", /* required */
  userId: 'USER_ID',          /* required */
  inputList: ["ITEM_ID_1", "ITEM_ID_2", "ITEM_ID_3", "ITEM_ID_4"]
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
    GetPersonalizedRankingCommand(getPersonalizedRankingParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [GetPersonalizedRanking](#) di Referensi AWS SDK for JavaScript API.

Dapatkan rekomendasi dari pemberi rekomendasi (grup dataset domain)

Contoh kode berikut menunjukkan cara mendapatkan rekomendasi Amazon Personalize Runtime Runtime.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";

import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',          /* required */
  numResults: 15             /* optional */
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
  GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Dapatkan rekomendasi dengan filter (grup kumpulan data khusus).

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  recommenderArn: 'RECOMMENDER_ARN', /* required */
```

```

    userId: 'USER_ID',      /* required */
    numResults: 15      /* optional */
  }

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
  GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();

```

Dapatkan rekomendasi yang difilter dari pemberi rekomendasi yang dibuat dalam grup kumpulan data domain.

```

// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here:
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',          /* required */
  numResults: 15,            /* optional */
  filterArn: 'FILTER_ARN',   /* required to filter recommendations */
  filterValues: {
    "PROPERTY": "\"VALUE\"" /* Only required if your filter has a placeholder
parameter */
  }
}

export const run = async () => {
  try {

```

```
const response = await personalizeRuntimeClient.send(new
GetRecommendationsCommand(getRecommendationsParam));
console.log("Success!", response);
return response; // For unit tests.
} catch (err) {
console.log("Error", err);
}
};
run();
```

- Untuk detail API, lihat [GetRecommendations](#) di Referensi AWS SDK for JavaScript API.

Amazon Pinpoint contoh menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon Pinpoint.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik


- [Tindakan](#)

Tindakan

Kirim email dan pesan teks

Contoh kode berikut menunjukkan cara mengirim email dan pesan teks dengan Amazon Pinpoint.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
//Set the MediaConvert Service Object
const pinClient = new PinpointClient({ region: REGION });
export { pinClient };
```

Kirim pesan email.

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

// The FromAddress must be verified in SES.
const fromAddress = "FROM_ADDRESS";
const toAddress = "TO_ADDRESS";
const projectId = "PINPOINT_PROJECT_ID";

// The subject line of the email.
var subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `
```

```
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
  using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;
```

```
// The character encoding for the subject line and message body of the email.
var charset = "UTF-8";
```

```
const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};
```

```
const run = async () => {
```



```
try {
  const data = await pinClient.send(new SendMessagesCommand(params));

  const {
    MessageResponse: { Result },
  } = data;

  const recipientResult = Result[toAddress];

  if (recipientResult.StatusCode !== 200) {
    throw new Error(recipientResult.StatusMessage);
  } else {
    console.log(recipientResult.MessageId);
  }
} catch (err) {
  console.log(err.message);
}
};

run();
```

Kirim pesan SMS.

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

("use strict");

/* The phone number or short code to send the message from. The phone number
or short code that you specify has to be associated with your Amazon Pinpoint
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number. For best results, you should specify the phone
number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
```

```
"using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
"opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderId";

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

const run = async () => {
```

```
try {
  const data = await pinClient.send(new SendMessagesCommand(params));
  return data; // For unit tests.
  console.log(
    "Message sent! " +
      data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
  );
} catch (err) {
  console.log(err);
}
};
run();
```

- Untuk detail API, lihat [SendMessages](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kirim pesan email.

```
"use strict";

const AWS = require("aws-sdk");

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
var toAddress = "recipient@example.com";
```

```
// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
```

```
ApplicationId: appId,
MessageRequest: {
  Addresses: {
    [toAddress]: {
      ChannelType: "EMAIL",
    },
  },
  MessageConfiguration: {
    EmailMessage: {
      FromAddress: senderAddress,
      SimpleEmail: {
        Subject: {
          Charset: charset,
          Data: subject,
        },
        HtmlPart: {
          Charset: charset,
          Data: body_html,
        },
        TextPart: {
          Charset: charset,
          Data: body_text,
        },
      },
    },
  },
},
},
};

//Try to send the email.
pinpoint.sendMessage(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
  } else {
    console.log(
      "Email sent! Message ID: ",
      data["MessageResponse"]["Result"][toAddress]["MessageId"]
    );
  }
});
```

Kirim pesan SMS.

```
"use strict";

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

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";

// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
var originationNumber = "+12065550199";

// The recipient's phone number. For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderId";
```

```
// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

//Try to send the message.
pinpoint.sendMessage(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
    // Otherwise, show the unique ID for the message.
  } else {
    console.log(
      "Message sent! " +
      data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  }
});
```

```
}  
});
```

- Untuk detail API, lihat [SendMessage](#) di Referensi AWS SDK for JavaScript API.

Contoh Amazon Redshift menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon Redshift.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Membuat klaster

Contoh kode berikut menunjukkan cara membuat cluster Amazon Redshift.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.


```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Buat cluster.

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

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least
  one uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
  your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
  cluster subnet group to be associated with this cluster. Defaults to 'default' if
  not specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Untuk detail API, lihat [CreateCluster](#) di Referensi AWS SDK for JavaScript API.

Hapus kluster

Contoh kode berikut menunjukkan cara menghapus cluster Amazon Redshift.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Buat cluster.

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

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
  }
}
```


```
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [DeleteCluster](#) di Referensi AWS SDK for JavaScript API.

Jelaskan cluster Anda

Contoh kode berikut menunjukkan cara mendeskripsikan cluster Amazon Redshift Anda.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Jelaskan cluster Anda.

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

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};
```

```
const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk detail API, lihat [DescribeClusters](#) di Referensi AWS SDK for JavaScript API.

Memodifikasi cluster

Contoh kode berikut menunjukkan cara memodifikasi cluster Amazon Redshift.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Memodifikasi cluster.

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

```
// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Untuk detail API, lihat [ModifyCluster](#) di Referensi AWS SDK for JavaScript API.

Contoh Amazon S3 menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon S3.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Amazon S3

Contoh kode berikut ini menunjukkan cara memulai menggunakan Amazon S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new S3Client({});

export const helloS3 = async () => {
  const command = new ListBucketsCommand({});

  const { Buckets } = await client.send(command);
  console.log("Buckets: ");
  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
  return Buckets;
};
```

- Untuk detail API, lihat [ListBuckets](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Tambahkan aturan CORS ke bucket

Contoh kode berikut menunjukkan cara menambahkan aturan cross-origin resource sharing (CORS) ke bucket S3.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Tambahkan aturan CORS.

```
import { PutBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// By default, Amazon S3 doesn't allow cross-origin requests. Use this command
// to explicitly allow cross-origin requests.
export const main = async () => {
  const command = new PutBucketCorsCommand({
    Bucket: "test-bucket",
    CORSConfiguration: {
      CORSRules: [
        {
          // Allow all headers to be sent to this bucket.
          AllowedHeaders: ["*"],
          // Allow only GET and PUT methods to be sent to this bucket.
          AllowedMethods: ["GET", "PUT"],
          // Allow only requests from the specified origin.
          AllowedOrigins: ["https://www.example.com"],
          // Allow the entity tag (ETag) header to be returned in the response. The
          ETag header
          // The entity tag represents a specific version of the object. The ETag
          reflects
          // changes only to the contents of an object, not its metadata.
          ExposeHeaders: ["ETag"],
          // How long the requesting browser should cache the preflight response.
          After
          // this time, the preflight request will have to be made again.
          MaxAgeSeconds: 3600,
        },
      ],
    },
  });
```

```
try {
  const response = await client.send(command);
  console.log(response);
} catch (err) {
  console.error(err);
}
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutBucketCors](#) di Referensi AWS SDK for JavaScript API.

Menambahkan kebijakan ke bucket

Contoh kode berikut menunjukkan cara menambahkan kebijakan ke bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Tambahkan kebijakan.

```
import { PutBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutBucketPolicyCommand({
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "AllowGetObject",
          // Allow this particular user to call GetObject on any object in this
          bucket.
          Effect: "Allow",
          Principal: {
```



```
        AWS: "arn:aws:iam::ACCOUNT-ID:user/USERNAME",
      },
      Action: "s3:GetObject",
      Resource: "arn:aws:s3:::BUCKET-NAME/*",
    },
  ],
}),
// Apply the preceding policy to this bucket.
Bucket: "BUCKET-NAME",
});

try {
  const response = await client.send(command);
  console.log(response);
} catch (err) {
  console.error(err);
}
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutBucketPolicy](#) di Referensi AWS SDK for JavaScript API.

Menyalin objek dari satu bucket ke bucket lainnya

Contoh kode berikut menunjukkan cara menyalin objek S3 dari satu ember ke ember lainnya.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Salin objek.

```
import { S3Client, CopyObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
```

```
const command = new CopyObjectCommand({
  CopySource: "SOURCE_BUCKET/SOURCE_OBJECT_KEY",
  Bucket: "DESTINATION_BUCKET",
  Key: "NEW_OBJECT_KEY",
});

try {
  const response = await client.send(command);
  console.log(response);
} catch (err) {
  console.error(err);
}
};
```

- Untuk detail API, lihat [CopyObject](#) di Referensi AWS SDK for JavaScript API.

Buat bucket

Contoh kode berikut menunjukkan cara membuat bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat bucket.

```
import { CreateBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CreateBucketCommand({
    // The name of the bucket. Bucket names are unique and have several other
    // constraints.
    // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
    bucketnamingrules.html
    Bucket: "bucket-name",
```

```
});

try {
  const { Location } = await client.send(command);
  console.log(`Bucket created with location ${Location}`);
} catch (err) {
  console.error(err);
}
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateBucket](#) di Referensi AWS SDK for JavaScript API.

Menghapus kebijakan dari bucket

Contoh kode berikut menunjukkan cara menghapus kebijakan dari bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus kebijakan bucket.

```
import { DeleteBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// This will remove the policy from the bucket.
export const main = async () => {
  const command = new DeleteBucketPolicyCommand({
    Bucket: "test-bucket",
  });

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

```
    } catch (err) {  
      console.error(err);  
    }  
  };  
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteBucketPolicy](#) di Referensi AWS SDK for JavaScript API.

Menghapus bucket kosong

Contoh kode berikut menunjukkan cara menghapus bucket S3 kosong.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus bucket.

```
import { DeleteBucketCommand, S3Client } from "@aws-sdk/client-s3";  
  
const client = new S3Client({});  
  
// Delete a bucket.  
export const main = async () => {  
  const command = new DeleteBucketCommand({  
    Bucket: "test-bucket",  
  });  
  
  try {  
    const response = await client.send(command);  
    console.log(response);  
  } catch (err) {  
    console.error(err);  
  }  
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteBucket](#) di Referensi AWS SDK for JavaScript API.

Menghapus objek

Contoh kode berikut menunjukkan cara menghapus objek S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh selengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus objek.

```
import { DeleteObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectCommand({
    Bucket: "test-bucket",
    Key: "test-key.txt",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DeleteObject](#) di Referensi AWS SDK for JavaScript API.

Menghapus beberapa objek

Contoh kode berikut menunjukkan cara menghapus beberapa objek dari bucket S3.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus beberapa objek.

```
import { DeleteObjectsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectsCommand({
    Bucket: "test-bucket",
    Delete: {
      Objects: [{ Key: "object1.txt" }, { Key: "object2.txt" }],
    },
  });

  try {
    const { Deleted } = await client.send(command);
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DeleteObjects](#) di Referensi AWS SDK for JavaScript API.

Menghapus konfigurasi situs web dari bucket

Contoh kode berikut menunjukkan cara menghapus konfigurasi situs web dari bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus konfigurasi situs web dari bucket.

```
import { DeleteBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Disable static website hosting on the bucket.
export const main = async () => {
  const command = new DeleteBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteBucketWebsite](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan aturan CORS untuk bucket

Contoh kode berikut menunjukkan cara mendapatkan aturan cross-origin resource sharing (CORS) untuk bucket S3.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan kebijakan CORS untuk bucket.

```
import { GetBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketCorsCommand({
    Bucket: "test-bucket",
  });

  try {
    const { CORSRules } = await client.send(command);
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `\n${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders.join(" ")}`,
        `\nAllowedMethods: ${cr.AllowedMethods.join(" ")}`,
        `\nAllowedOrigins: ${cr.AllowedOrigins.join(" ")}`,
        `\nExposeHeaders: ${cr.ExposeHeaders.join(" ")}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetBucketCors](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan objek dari bucket

Contoh kode berikut menunjukkan cara membaca data dari objek dalam bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Unduh objek tersebut.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
  });

  try {
    const response = await client.send(command);
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetObject](#) di Referensi AWS SDK for JavaScript API.

Mendapatkan ACL dari bucket

Contoh kode berikut menunjukkan cara mendapatkan daftar kontrol akses (ACL) dari bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan izin ACL.

```
import { GetBucketAclCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketAclCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetBucketAcl](#) di Referensi AWS SDK for JavaScript API.

Dapatkan kebijakan untuk bucket

Contoh kode berikut menunjukkan cara mendapatkan kebijakan untuk bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan kebijakan bucket.

```
import { GetBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const { Policy } = await client.send(command);
    console.log(JSON.parse(Policy));
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetBucketPolicy](#) di Referensi AWS SDK for JavaScript API.

Dapatkan konfigurasi situs web untuk bucket

Contoh kode berikut menunjukkan cara mendapatkan konfigurasi situs web untuk bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan konfigurasi situs web.

```
import { GetBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const { ErrorDocument, IndexDocument } = await client.send(command);
    console.log(
      `Your bucket is set up to host a website. It has an error document:`,
      `${ErrorDocument.Key}, and an index document: ${IndexDocument.Suffix}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [GetBucketWebsite](#) di Referensi AWS SDK for JavaScript API.

Buat daftar bucket

Contoh kode berikut menunjukkan cara membuat daftar bucket S3.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar bucket.

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

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

  try {
    const { Owner, Buckets } = await client.send(command);
    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`,
    );
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListBuckets](#) di Referensi AWS SDK for JavaScript API.

Membuat daftar objek dalam bucket

Contoh kode berikut menunjukkan cara membuat daftar objek dalam bucket S3.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya, dan pelajari cara untuk mengatur dan menjalankannya di [AWS Repositori Contoh Kode](#).

Mencantumkan semua objek di dalam bucket Anda. Jika ada lebih dari satu objek, `IsTruncated` dan `NextContinuationToken` akan digunakan untuk iterasi atas daftar lengkap.

```
import {
  S3Client,
  // This command supersedes the ListObjectsCommand and is the recommended way to
  // list objects.
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListObjectsV2Command({
    Bucket: "my-bucket",
    // The default and maximum number of keys returned is 1000. This limits it to
    // one for demonstration purposes.
    MaxKeys: 1,
  });

  try {
    let isTruncated = true;

    console.log("Your bucket contains the following objects:\n");
    let contents = "";

    while (isTruncated) {
      const { Contents, IsTruncated, NextContinuationToken } =
        await client.send(command);
      const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
      contents += contentsList + "\n";
      isTruncated = IsTruncated;
      command.input.ContinuationToken = NextContinuationToken;
    }
  }
}
```

```
    console.log(contents);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [ListObjectsV2](#) di Referensi AWS SDK for JavaScript API.

Menetapkan ACL baru untuk bucket

Contoh kode berikut menunjukkan cara mengatur daftar kontrol akses (ACL) baru untuk bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Tempatkan bucket ACL.

```
import {
  PutBucketAclCommand,
  GetBucketAclCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const client = new S3Client({});

// Most Amazon S3 use cases don't require the use of access control lists (ACLs).
// We recommend that you disable ACLs, except in unusual circumstances where
// you need to control access for each object individually.
// Consider a policy instead. For more information see https://docs.aws.amazon.com/
// AmazonS3/latest/userguide/bucket-policies.html.
export const main = async () => {
  // Grant a user READ access to a bucket.
  const command = new PutBucketAclCommand({
    Bucket: "test-bucket",
    AccessControlPolicy: {
```

```

    Grants: [
      {
        Grantee: {
          // The canonical ID of the user. This ID is an obfuscated form of your
          AWS account number.
          // It's unique to Amazon S3 and can't be found elsewhere.
          // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
          ID: "canonical-id-1",
          Type: "CanonicalUser",
        },
        // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
        // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
        Permission: "FULL_CONTROL",
      },
    ],
    Owner: {
      ID: "canonical-id-2",
    },
  },
});

try {
  const response = await client.send(command);
  console.log(response);
} catch (err) {
  console.error(err);
}
};


```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutBucketAcl](#) di Referensi AWS SDK for JavaScript API.

Menetapkan konfigurasi situs web untuk bucket

Contoh kode berikut menunjukkan cara mengatur konfigurasi situs web untuk bucket S3.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Atur konfigurasi situs web.

```
import { PutBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Set up a bucket as a static website.
// The bucket needs to be publicly accessible.
export const main = async () => {
  const command = new PutBucketWebsiteCommand({
    Bucket: "test-bucket",
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
        // A suffix that is appended to a request that is for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutBucketWebsite](#) di Referensi AWS SDK for JavaScript API.

Mengunggah objek ke bucket

Contoh kode berikut menunjukkan cara mengunggah objek ke bucket S3.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Unggah objek tersebut.

```
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
    Body: "Hello S3!",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutObject](#) di Referensi AWS SDK for JavaScript API.

Skenario

Membuat URL yang telah ditetapkan sebelumnya

Contoh kode berikut menunjukkan cara membuat URL presigned untuk Amazon S3 dan mengunggah objek.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat URL yang telah ditetapkan sebelumnya untuk unggah objek ke bucket.

```
import https from "https";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
  return formatUrl(signedUrlObject);
};
```

```
const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

function put(url, data) {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          resolve(responseBody);
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
}

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.txt";

  // There are two ways to generate a presigned URL.
  // 1. Use createPresignedUrl without the S3 client.
  // 2. Use getSignedUrl in conjunction with the S3 client and GetObjectCommand.
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });
  }
}
```

```

const clientUrl = await createPresignedUrlWithClient({
  region: REGION,
  bucket: BUCKET,
  key: KEY,
});

// After you get the presigned URL, you can provide your own file
// data. Refer to put() above.
console.log("Calling PUT using presigned URL without client");
await put(noClientUrl, "Hello World");

console.log("Calling PUT using presigned URL with client");
await put(clientUrl, "Hello World");

console.log("\nDone. Check your S3 console.");
} catch (err) {
  console.error(err);
}
};

```

Buat URL yang telah ditetapkan sebelumnya untuk mengunduh objek dari bucket.

```

import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(new HttpRequest(url));

```

```
    return formatUrl(signedUrlObject);
  };

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new GetObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.jpg";

  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    console.log("Presigned URL without client");
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk informasi selengkapnya, lihat [Panduan Developer AWS SDK for JavaScript](#).

Membuat halaman web yang mencantumkan objek Amazon S3

Contoh kode berikut menunjukkan cara mendaftar objek Amazon S3 di halaman web.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kode berikut adalah komponen React yang relevan yang membuat panggilan ke AWS SDK. Versi runnable dari aplikasi yang berisi komponen ini dapat ditemukan di link sebelumnya GitHub .

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";

function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
      // role to the pool,
      // and grant the role access to the 's3:GetObject' action.
      //
      // You'll also need to configure the CORS settings on the bucket to allow
      // traffic from
      // this example site. Here's an example configuration that allows all origins.
      // Don't
      // do this in production.
    });
  });
}
```

```
//[
// {
//   "AllowedHeaders": ["*"],
//   "AllowedMethods": ["GET"],
//   "AllowedOrigins": ["*"],
//   "ExposeHeaders": [],
// },
//]
//
credentials: fromCognitoIdentityPool({
  clientConfig: { region: "us-east-1" },
  identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
}),
});
const command = new ListObjectsCommand({ Bucket: "bucket-name" });
client.send(command).then(({ Contents }) => setObjects(Contents || []));
}, []);

return (
  <div className="App">
    {objects.map((o) => (
      <div key={o.ETag}>{o.Key}</div>
    ))}
  </div>
);
}

export default App;
```

- Untuk detail API, lihat [ListObjects](#) di Referensi AWS SDK for JavaScript API.

Memulai bucket dan objek

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Membuat bucket dan mengunggah file ke dalamnya.
- Mengunduh objek dari bucket.
- Menyalin objek ke subfolder di bucket.
- Membuat daftar objek dalam bucket.
- Menghapus objek bucket dan bucket tersebut.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Pertama, impor semua modul yang diperlukan.

```
// Used to check if currently running file is this file.
import { fileURLToPath } from "url";
import { readdirSync, readFileSync, writeFileSync } from "fs";

// Local helper utils.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

import {
  S3Client,
  CreateBucketCommand,
  PutObjectCommand,
  ListObjectsCommand,
  CopyObjectCommand,
  GetObjectCommand,
  DeleteObjectsCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
```

Impor sebelumnya merujuk beberapa utilitas pembantu. Utilitas ini bersifat lokal ke GitHub repositori yang ditautkan di awal bagian ini. Untuk referensi, lihat implementasi utilitas tersebut berikut ini.

```
export const dirnameFromMetaUrl = (metaUrl) =>
  fileURLToPath(new URL(".", metaUrl));

import { select, input, confirm, checkbox } from "@inquirer/prompts";

export class Prompter {
  /**
```

```
* @param {{ message: string, choices: { name: string, value: string }[]}} options
*/
select(options) {
  return select(options);
}

/**
 * @param {{ message: string }} options
 */
input(options) {
  return input(options);
}

/**
 * @param {string} prompt
 */
checkContinue = async (prompt = "") => {
  const prefix = prompt && prompt + " ";
  let ok = await this.confirm({
    message: `${prefix}Continue?`,
  });
  if (!ok) throw new Error("Exiting...");
};

/**
 * @param {{ message: string }} options
 */
confirm(options) {
  return confirm(options);
}

/**
 * @param {{ message: string, choices: { name: string, value: string }[]}} options
 */
checkbox(options) {
  return checkbox(options);
}
}

export const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n  ${text}\n${rule}\n`;
};
```

Objek di S3 disimpan dalam 'bucket'. Mari kita tentukan fungsi untuk membuat bucket baru.

```
export const createBucket = async () => {
  const bucketName = await prompter.input({
    message: "Enter a bucket name. Bucket names must be globally unique:",
  });
  const command = new CreateBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log("Bucket created successfully.\n");
  return bucketName;
};
```

Bucket berisi 'objek'. Fungsi ini mengunggah isi direktori ke bucket Anda sebagai objek.

```
export const uploadFilesToBucket = async ({ bucketName, folderPath }) => {
  console.log(`Uploading files from ${folderPath}\n`);
  const keys = readdirSync(folderPath);
  const files = keys.map((key) => {
    const filePath = `${folderPath}/${key}`;
    const fileContent = readFileSync(filePath);
    return {
      Key: key,
      Body: fileContent,
    };
  });

  for (let file of files) {
    await s3Client.send(
      new PutObjectCommand({
        Bucket: bucketName,
        Body: file.Body,
        Key: file.Key,
      })
    );
    console.log(`${file.Key} uploaded successfully.`);
  }
};
```

Setelah mengunggah objek, pastikan bahwa objek tersebut diunggah dengan benar. Anda dapat menggunakannya ListObjects untuk itu. Anda akan menggunakan properti 'Kunci', tetapi ada juga properti lain yang berguna dalam respons.

```
export const listFilesInBucket = async ({ bucketName }) => {
  const command = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(command);
  const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
  console.log("\nHere's a list of files in the bucket:");
  console.log(contentsList + "\n");
};
```

Terkadang Anda perlu menyalin objek dari satu bucket ke bucket lainnya. Gunakan CopyObject perintah untuk itu.

```
export const copyFileFromBucket = async ({ destinationBucket }) => {
  const proceed = await prompter.confirm({
    message: "Would you like to copy an object from another bucket?",
  });

  if (!proceed) {
    return;
  } else {
    const copy = async () => {
      try {
        const sourceBucket = await prompter.input({
          message: "Enter source bucket name:",
        });
        const sourceKey = await prompter.input({
          message: "Enter source key:",
        });
        const destinationKey = await prompter.input({
          message: "Enter destination key:",
        });

        const command = new CopyObjectCommand({
          Bucket: destinationBucket,
          CopySource: `${sourceBucket}/${sourceKey}`,
          Key: destinationKey,
        });
        await s3Client.send(command);
        await copyFileFromBucket({ destinationBucket });
      } catch (error) {
        console.error(error);
      }
    };
  }
};
```

```
    } catch (err) {
      console.error(`Copy error.`);
      console.error(err);
      const retryAnswer = await prompter.confirm({ message: "Try again?" });
      if (retryAnswer) {
        await copy();
      }
    }
  };
  await copy();
}
};
```

Tidak ada metode SDK yang bisa digunakan untuk mendapatkan beberapa objek dari sebuah bucket. Tetapi, Anda akan membuat daftar objek yang akan diunduh dan mengulanginya.

```
export const downloadFilesFromBucket = async ({ bucketName }) => {
  const { Contents } = await s3Client.send(
    new ListObjectsCommand({ Bucket: bucketName }),
  );
  const path = await prompter.input({
    message: "Enter destination path for files:",
  });

  for (let content of Contents) {
    const obj = await s3Client.send(
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
    );
    writeFileSync(
      `${path}/${content.Key}`,
      await obj.Body.transformToByteArray(),
    );
  }
  console.log("Files downloaded successfully.\n");
};
```

Saatnya membersihkan sumber daya Anda. Bucket harus kosong sebelum dapat dihapus. Kedua fungsi ini mengosongkan dan menghapus bucket.

```
export const emptyBucket = async ({ bucketName }) => {
```

```

const listObjectsCommand = new ListObjectsCommand({ Bucket: bucketName });
const { Contents } = await s3Client.send(listObjectsCommand);
const keys = Contents.map((c) => c.Key);

const deleteObjectsCommand = new DeleteObjectsCommand({
  Bucket: bucketName,
  Delete: { Objects: keys.map((key) => ({ Key: key })) },
});
await s3Client.send(deleteObjectsCommand);
console.log(`${bucketName} emptied successfully.\n`);
};

export const deleteBucket = async ({ bucketName }) => {
  const command = new DeleteBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log(`${bucketName} deleted successfully.\n`);
};

```

Fungsi 'utama' menyatukan semuanya. Jika Anda menjalankan file ini secara langsung, fungsi utama akan dipanggil.

```

const main = async () => {
  const OBJECT_DIRECTORY = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../resources/sample_files/.sample_media`;

  try {
    console.log(wrapText("Welcome to the Amazon S3 getting started example."));
    console.log("Let's create a bucket.");
    const bucketName = await createBucket();
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("File upload."));
    console.log(
      "I have some default files ready to go. You can edit the source code to provide your own.",
    );
    await uploadFilesToBucket({
      bucketName,
      folderPath: OBJECT_DIRECTORY,
    });
  }
};

```

```
    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Copy files."));
    await copyFileFromBucket({ destinationBucket: bucketName });
    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Download files."));
    await downloadFilesFromBucket({ bucketName });

    console.log(wrapText("Clean up."));
    await emptyBucket({ bucketName });
    await deleteBucket({ bucketName });
  } catch (err) {
    console.error(err);
  }
};
```


- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for JavaScript .
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjectsV2](#)
 - [PutObject](#)

Mengunggah atau mengunduh file besar

Contoh kode berikut menunjukkan cara mengunggah atau mengunduh file besar ke dan dari Amazon S3.

Untuk informasi selengkapnya, lihat [Pengunggahan objek menggunakan unggahan multibagian](#).

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Unggah file besar.

```
import {
  CreateMultipartUploadCommand,
  UploadPartCommand,
  CompleteMultipartUploadCommand,
  AbortMultipartUploadCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

export const main = async () => {
  const s3Client = new S3Client({});
  const bucketName = "test-bucket";
  const key = "multipart.txt";
  const str = createString();
  const buffer = Buffer.from(str, "utf8");

  let uploadId;

  try {
    const multipartUpload = await s3Client.send(
      new CreateMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );

    uploadId = multipartUpload.UploadId;
```



```
const uploadPromises = [];
// Multipart uploads require a minimum size of 5 MB per part.
const partSize = Math.ceil(buffer.length / 5);

// Upload each part.
for (let i = 0; i < 5; i++) {
  const start = i * partSize;
  const end = start + partSize;
  uploadPromises.push(
    s3Client
      .send(
        new UploadPartCommand({
          Bucket: bucketName,
          Key: key,
          UploadId: uploadId,
          Body: buffer.subarray(start, end),
          PartNumber: i + 1,
        })
      )
      .then((d) => {
        console.log("Part", i + 1, "uploaded");
        return d;
      })
  );
}

const uploadResults = await Promise.all(uploadPromises);

return await s3Client.send(
  new CompleteMultipartUploadCommand({
    Bucket: bucketName,
    Key: key,
    UploadId: uploadId,
    MultipartUpload: {
      Parts: uploadResults.map(({ ETag }, i) => ({
        ETag,
        PartNumber: i + 1,
      })),
    },
  })
);

// Verify the output by downloading the file from the Amazon Simple Storage
Service (Amazon S3) console.
```

```
// Because the output is a 25 MB string, text editors might struggle to open the
file.
} catch (err) {
  console.error(err);

  if (uploadId) {
    const abortCommand = new AbortMultipartUploadCommand({
      Bucket: bucketName,
      Key: key,
      UploadId: uploadId,
    });

    await s3Client.send(abortCommand);
  }
}
};
```

Unggah file besar.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream } from "fs";

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;

export const getObjectRange = ({ bucket, key, start, end }) => {
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: parseInt(start),
    end: parseInt(end),
    length: parseInt(length),
  };
};
```

```
};
};

export const isComplete = ({ end, length }) => end === length - 1;

// When downloading a large file, you might want to break it down into
// smaller pieces. Amazon S3 accepts a Range header to specify the start
// and end of the byte range to be downloaded.
const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url))
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    console.log(`Downloading bytes ${nextRange.start} to ${nextRange.end}`);

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
    });

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

export const main = async () => {
  await downloadInChunks({
    bucket: "my-cool-bucket",
    key: "my-cool-object.txt",
  });
};
```

Contoh Gletser S3 menggunakan SDK untuk (v3) JavaScript

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan S3 Glacier.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Buat lemari besi

Contoh kode berikut menunjukkan cara membuat lemari besi Amazon S3 Glacier.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Buat lemari besi.

```
// Load the SDK for JavaScript
import { CreateVaultCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "../libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME
const params = { vaultName: vaultname };

const run = async () => {
  try {
    const data = await glacierClient.send(new CreateVaultCommand(params));
    console.log("Success, vault created!");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error");
  }
};
run();
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateVault](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create a new service object
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
```

```
// Call Glacier to create the vault
glacier.createVault({ vaultName: "YOUR_VAULT_NAME" }, function (err) {
  if (!err) {
    console.log("Created vault!");
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateVault](#) di Referensi AWS SDK for JavaScript API.

Unggah arsip ke brankas

Contoh kode berikut menunjukkan cara mengunggah arsip ke brankas Amazon S3 Glacier.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Unggah arsip.

```
// Load the SDK for JavaScript
import { UploadArchiveCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "../libs/glacierClient.js";

// Set the parameters
```

```
const vaultname = "VAULT_NAME"; // VAULT_NAME

// Create a new service object and buffer
const buffer = new Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer
const params = { vaultName: vaultname, body: buffer };

const run = async () => {
  try {
    const data = await glacierClient.send(new UploadArchiveCommand(params));
    console.log("Archive ID", data.archiveId);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error uploading archive!", err);
  }
};
run();
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [UploadArchive](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

// Create a new service object and buffer
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
buffer = Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer

var params = { vaultName: "YOUR_VAULT_NAME", body: buffer };
// Call Glacier to upload the archive.
glacier.uploadArchive(params, function (err, data) {
  if (err) {
```

```
    console.log("Error uploading archive!", err);
  } else {
    console.log("Archive ID", data.archiveId);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [UploadArchive](#) di Referensi AWS SDK for JavaScript API.

SageMaker contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) with SageMaker.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo SageMaker

Contoh kode berikut menunjukkan cara untuk mulai menggunakan SageMaker.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import {
```



```
SageMakerClient,  
ListNotebookInstancesCommand,  
} from "@aws-sdk/client-sagemaker";  
  
const client = new SageMakerClient({  
  region: "us-west-2",  
});  
  
export const helloSagemaker = async () => {  
  const command = new ListNotebookInstancesCommand({ MaxResults: 5 });  
  
  const response = await client.send(command);  
  console.log(  
    "Hello Amazon SageMaker! Let's list some of your notebook instances:",  
  );  
  
  const instances = response.NotebookInstances || [];  
  
  if (instances.length === 0) {  
    console.log(  
      "• No notebook instances found. Try creating one in the AWS Management Console  
or with the CreateNotebookInstanceCommand.",  
    );  
  } else {  
    console.log(  
      instances  
        .map(  
          (i) =>  
            `• Instance: ${i.NotebookInstanceName}\n  Arn:${  
              i.NotebookInstanceArn  
            } \n  Creation Date: ${i.CreationTime.toISOString()}`,  
        )  
        .join("\n"),  
    );  
  }  
  
  return response;  
};
```

- Untuk detail API, lihat [ListNotebookInstances](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Buat pipeline

Contoh kode berikut menunjukkan cara membuat atau memperbarui pipeline di SageMaker.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Fungsi yang membuat SageMaker pipeline menggunakan definisi JSON yang disediakan secara lokal.

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 * definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
 * sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
    implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
```

```
    )}../../../../../../../../workflows/sagemaker_pipelines/resources/  
GeoSpatialPipeline.json`,  
  )  
  .toString()  
  .replace(/.*FUNCTION_ARN*/g, functionArn);  
  
const { PipelineArn } = await sagemakerClient.send(  
  new CreatePipelineCommand({  
    PipelineName: name,  
    PipelineDefinition: pipelineDefinition,  
    RoleArn: roleArn,  
  })),  
);  
  
return {  
  arn: PipelineArn,  
  cleanUp: async () => {  
    await sagemakerClient.send(  
      new DeletePipelineCommand({ PipelineName: name })),  
    );  
  },  
};  
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [CreatePipeline](#)
 - [UpdatePipeline](#)

Hapus pipa

Contoh kode berikut menunjukkan cara menghapus pipeline di SageMaker.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Sintaks untuk menghapus pipeline. SageMaker Kode ini adalah bagian dari fungsi yang lebih besar. Lihat 'Buat saluran pipa' atau GitHub repositori untuk konteks lebih lanjut.

```
await sagemakerClient.send(  
  new DeletePipelineCommand({ PipelineName: name } ),  
);
```

- Untuk detail API, lihat [DeletePipeline](#) di Referensi AWS SDK for JavaScript API.

Jelaskan eksekusi pipeline

Contoh kode berikut menunjukkan cara mendeskripsikan eksekusi pipeline di SageMaker.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Tunggu eksekusi SageMaker pipeline berhasil, gagal, atau berhenti.

```
/**  
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or  
 * 'FAILED'.  
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-  
sagemaker').SageMakerClient}} props  
 */  
export async function waitForPipelineComplete({ arn, sagemakerClient }) {  
  const command = new DescribePipelineExecutionCommand({  
    PipelineExecutionArn: arn,  
  });  
  
  let complete = false;  
  let intervalInSeconds = 15;  
  const COMPLETION_STATUSES = [  
    PipelineExecutionStatus.FAILED,  
    PipelineExecutionStatus.STOPPED,  
    PipelineExecutionStatus.SUCCEEDED,
```

```
];

do {
  const { PipelineExecutionStatus: status, FailureReason } =
    await sagemakerClient.send(command);

  complete = COMPLETION_STATUSES.includes(status);

  if (!complete) {
    console.log(
      `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
again.`
    );
    await wait(intervalInSeconds);
  } else if (status === PipelineExecutionStatus.FAILED) {
    throw new Error(`Pipeline failed because: ${FailureReason}`);
  } else if (status === PipelineExecutionStatus.STOPPED) {
    throw new Error(`Pipeline was forcefully stopped.`);
  } else {
    console.log(`Pipeline execution ${status}.`);
  }
} while (!complete);
}
```

- Untuk detail API, lihat [DescribePipelineExecution](#) di Referensi AWS SDK for JavaScript API.

Jalankan pipa

Contoh kode berikut menunjukkan cara memulai eksekusi pipeline di SageMaker.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mulai eksekusi SageMaker pipeline.

```
/**
```

```

* Start the execution of the Amazon SageMaker pipeline. Parameters that are
* passed in are used in the AWS Lambda function.
* @param {{
*   name: string,
*   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
*   roleArn: string,
*   queueUrl: string,
*   s3InputBucketName: string,
* }} props
*/
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
   * configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };
};

```

```
/**
 * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
 * latitude and longitude values.
 * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
 */
const jobConfig = {
  ReverseGeocodingConfig: {
    XAttributeName: "Longitude",
    YAttributeName: "Latitude",
  },
};

const { PipelineExecutionArn } = await sagemakerClient.send(
  new StartPipelineExecutionCommand({
    PipelineName: name,
    PipelineExecutionDisplayName: `${name}-example-execution`,
    PipelineParameters: [
      { Name: "parameter_execution_role", Value: roleArn },
      { Name: "parameter_queue_url", Value: queueUrl },
      {
        Name: "parameter_vej_input_config",
        Value: JSON.stringify(inputConfig),
      },
      {
        Name: "parameter_vej_export_config",
        Value: JSON.stringify(outputConfig),
      },
      {
        Name: "parameter_step_1_vej_config",
        Value: JSON.stringify(jobConfig),
      },
    ],
  }),
);

return {
  arn: PipelineExecutionArn,
};
}
```

- Untuk detail API, lihat [StartPipelineExecution](#) di Referensi AWS SDK for JavaScript API.

Skenario

Memulai pekerjaan geospasial dan jaringan pipa

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Siapkan sumber daya untuk pipa.
- Siapkan pipa yang menjalankan pekerjaan geospasial.
- Mulai eksekusi pipeline.
- Pantau status eksekusi.
- Lihat output dari pipa.
- Pembersihan sumber daya

Untuk informasi selengkapnya, lihat [Membuat dan menjalankan SageMaker pipeline menggunakan AWS SDK di Community.aws](#).

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kutipan file berikut berisi fungsi yang menggunakan SageMaker klien untuk mengelola pipeline.

```
import { readFileSync } from "fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
} from "@aws-sdk/client-iam";
```



```
import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry, wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 * props
 */
```

```

*/
export async function createLambdaExecutionRole({ name, iamClient }) {
  const { Role } = await iamClient.send(
    new CreateRoleCommand({
      RoleName: name,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Action: ["sts:AssumeRole"],
            Principal: { Service: ["lambda.amazonaws.com"] },
          },
        ],
      })),
  );

  return {
    arn: Role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 * AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 * Amazon SageMaker.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 * pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policy = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",

```

```

    Action: [
      "sqs:ReceiveMessage",
      "sqs>DeleteMessage",
      "sqs:GetQueueAttributes",
      "logs>CreateLogGroup",
      "logs>CreateLogStream",
      "logs:PutLogEvents",
      "sagemaker-geospatial:StartVectorEnrichmentJob",
      "sagemaker-geospatial:GetVectorEnrichmentJob",
      "sagemaker:SendPipelineExecutionStepFailure",
      "sagemaker:SendPipelineExecutionStepSuccess",
      "sagemaker-geospatial:ExportVectorEnrichmentJob",
    ],
    Resource: "*",
  },
  {
    Effect: "Allow",
    // The AWS Lambda function needs permission to pass the pipeline execution
role to
    // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
function
    // from elevating privileges. For more information, see:
    // https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_use_passrole.html
    Action: ["iam:PassRole"],
    Resource: `${pipelineExecutionRoleArn}`,
    Condition: {
      StringEquals: {
        "iam:PassedToService": [
          "sagemaker.amazonaws.com",
          "sagemaker-geospatial.amazonaws.com",
        ],
      },
    },
  },
],
};

const createPolicyCommand = new CreatePolicyCommand({
  PolicyDocument: JSON.stringify(policy),
  PolicyName: name,
});

const { Policy } = await iamClient.send(createPolicyCommand);

```

```

    return {
      arn: Policy.Arn,
      policy,
      cleanUp: async () => {
        await iamClient.send(new DeletePolicyCommand({ PolicyArn: Policy.Arn }));
      },
    };
  }

  /**
   * Attach an AWS IAM policy to an AWS IAM role.
   * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-iam').IAMClient}} props
   */
  export async function attachPolicy({ roleName, policyArn, iamClient }) {
    const attachPolicyCommand = new AttachRolePolicyCommand({
      RoleName: roleName,
      PolicyArn: policyArn,
    });

    await iamClient.send(attachPolicyCommand);
    return {
      cleanUp: async () => {
        await iamClient.send(
          new DetachRolePolicyCommand({
            RoleName: roleName,
            PolicyArn: policyArn,
          }),
        );
      },
    };
  }

  /**
   * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon
   SageMaker Geospatial clients
   * in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
   Amazon SageMaker
   * Geospatial client wasn't introduced until v3.221.0.
   * @param {{ name: string, lambdaClient: import('@aws-sdk/client-lambda').LambdaClient }} props
   */
  export async function createLambdaLayer({ name, lambdaClient }) {
    const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;

```

```
const { LayerVersionArn, Version } = await lambdaClient.send(
  new PublishLayerVersionCommand({
    LayerName: name,
    Content: {
      ZipFile: Uint8Array.from(readFileSync(layerPath)),
    },
  }),
);

return {
  versionArn: LayerVersionArn,
  version: Version,
  cleanUp: async () => {
    await lambdaClient.send(
      new DeleteLayerVersionCommand({
        LayerName: name,
        VersionNumber: Version,
      }),
    );
  },
};
}

/**
 * Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
 * pipeline
 * execution steps.
 * @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
 * lambda').LambdaClient, layerVersionArn: string}} props
 */
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
  layerVersionArn,
}) {
  const lambdaPath = `${dirnameFromMetaUrl(
    import.meta.url,
  )}lambda/dist/index.mjs.zip`;

  const command = new CreateFunctionCommand({
    Code: {
      ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
    },
  });
}
```

```
    Runtime: Runtime.nodejs18x,  
    Handler: "index.handler",  
    Layers: [layerVersionArn],  
    FunctionName: name,  
    Role: roleArn,  
  });  
  
  // Function creation fails if the Role is not ready. This retries  
  // function creation until it succeeds or it times out.  
  const { FunctionArn } = await retry(  
    { intervalInMs: 1000, maxRetries: 60 },  
    () => lambdaClient.send(command),  
  );  
  
  return {  
    arn: FunctionArn,  
    cleanUp: async () => {  
      await lambdaClient.send(  
        new DeleteFunctionCommand({ FunctionName: name } ),  
      );  
    },  
  };  
}  
  
/**  
 * This uploads some sample coordinate data to an Amazon S3 bucket.  
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/  
Long  
 * coordinates in this file and augment them with more detailed location data.  
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}  
props  
 */  
export async function uploadCSVDataToS3({ bucketName, s3Client }) {  
  const s3Path = `${dirnameFromMetaUrl(  
    import.meta.url,  
  )}../../../../../../../../workflows/sagemaker_pipelines/resources/latlongtest.csv`;  
  
  await s3Client.send(  
    new PutObjectCommand({  
      Bucket: bucketName,  
      Key: "input/sample_data.csv",  
      Body: readFileSync(s3Path),  
    } ),  
  );  
}
```

```
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient}} props
 */
export async function createSagemakerRole({ name, iamClient }) {
  const command = new CreateRoleCommand({
    RoleName: name,
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["sts:AssumeRole"],
          Principal: {
            Service: [
              "sagemaker.amazonaws.com",
              "sagemaker-geospatial.amazonaws.com",
            ],
          },
        },
      ],
    }),
  });

  const { Role } = await iamClient.send(command);
  // Wait for the role to be ready.
  await wait(10);

  return {
    arn: Role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 * messages to
 * the Amazon SQS queue.
```

```
* @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
*/
export async function createSagemakerExecutionPolicy({
  sqsQueueArn,
  lambdaArn,
  iamClient,
  name,
  s3BucketName,
}) {
  const policy = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: ["lambda:InvokeFunction"],
        Resource: lambdaArn,
      },
      {
        Effect: "Allow",
        Action: ["s3:*"],
        Resource: [
          `arn:aws:s3:::${s3BucketName}`,
          `arn:aws:s3:::${s3BucketName}/*`,
        ],
      },
      {
        Effect: "Allow",
        Action: ["sqs:SendMessage"],
        Resource: sqsQueueArn,
      },
    ],
  };

  const createPolicyCommand = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify(policy),
    PolicyName: name,
  });

  const { Policy } = await iamClient.send(createPolicyCommand);
  return {
    arn: Policy.Arn,
    policy,
    cleanup: async () => {
```



```
    await iamClient.send(new DeletePolicyCommand({ PolicyArn: Policy.Arn }));
  },
};
}

/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 * definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
 * sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
    implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../workflows/sagemaker_pipelines/resources/
    GeoSpatialPipeline.json`,
  )
  .toString()
  .replace(/.*FUNCTION_ARN.*/g, functionArn);

  const { PipelineArn } = await sagemakerClient.send(
    new CreatePipelineCommand({
      PipelineName: name,
      PipelineDefinition: pipelineDefinition,
      RoleArn: roleArn,
    }),
  );

  return {
    arn: PipelineArn,
    cleanUp: async () => {
      await sagemakerClient.send(
```

```
        new DeletePipelineCommand({ PipelineName: name }),
    );
  },
};
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
  const { QueueUrl } = await sqsClient.send(
    new CreateQueueCommand({
      QueueName: name,
      Attributes: {
        DelaySeconds: "5",
        ReceiveMessageWaitTimeSeconds: "5",
        VisibilityTimeout: "300",
      },
    }),
  );

  const { Attributes } = await sqsClient.send(
    new GetQueueAttributesCommand({
      QueueUrl,
      AttributeNames: ["QueueArn"],
    }),
  );

  return {
    queueUrl: QueueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.

```

```
* @param {{lambdaName: string, queueArn: string, lambdaClient: import('@aws-sdk/
client-lambda').LambdaClient, sqsClient: import('@aws-sdk/client-sqs').SQSClient}}
props
*/
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
}) {
  const { UUID } = await lambdaClient.send(
    new CreateEventSourceMappingCommand({
      EventSourceArn: queueArn,
      FunctionName: lambdaName,
    }),
  );

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
          UUID,
        }),
      );
    },
  };
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{s3Client: import('@aws-sdk/client-s3').S3Client, name: string}} props
 */
export async function createS3Bucket({ name, s3Client }) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(
        { client: s3Client },
        { Bucket: name },
      );
      for await (const page of paginator) {
        const objects = page.Contents;
        if (objects) {
```

```

        for (const object of objects) {
            await s3Client.send(
                new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
            );
        }
    }
    await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
},
];
}

/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
    sagemakerClient,
    name,
    bucketName,
    roleArn,
    queueUrl,
}) {
    /**
     * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
     * file in an Amazon S3 bucket.
     * @type {import("@aws-sdk/client-sagemaker-geospatial").VectorEnrichmentJobInputConfig}
     */
    const inputConfig = {
        DataSourceConfig: {
            S3Data: {
                S3Uri: `s3://${bucketName}/input/sample_data.csv`,
            },
        },
        DocumentType: VectorEnrichmentJobDocumentType.CSV,
    };
}

```

```
/**
 * The Vector Enrichment Job adds additional data to the source CSV. This
configuration points
 * to an Amazon S3 prefix where the output will be stored.
 * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
 */
const outputConfig = {
  S3Data: {
    S3Uri: `s3://${bucketName}/output/`,
  },
};

/**
 * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
requires
 * latitude and longitude values.
 * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
 */
const jobConfig = {
  ReverseGeocodingConfig: {
    XAttributeName: "Longitude",
    YAttributeName: "Latitude",
  },
};

const { PipelineExecutionArn } = await sagemakerClient.send(
  new StartPipelineExecutionCommand({
    PipelineName: name,
    PipelineExecutionDisplayName: `${name}-example-execution`,
    PipelineParameters: [
      { Name: "parameter_execution_role", Value: roleArn },
      { Name: "parameter_queue_url", Value: queueUrl },
      {
        Name: "parameter_vej_input_config",
        Value: JSON.stringify(inputConfig),
      },
      {
        Name: "parameter_vej_export_config",
        Value: JSON.stringify(outputConfig),
      },
    ],
  })
);
```

```
        Name: "parameter_step_1_vej_config",
        Value: JSON.stringify(jobConfig),
    },
],
}),
);

return {
    arn: PipelineExecutionArn,
};
}

/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 * 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient }} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient }) {
    const command = new DescribePipelineExecutionCommand({
        PipelineExecutionArn: arn,
    });

    let complete = false;
    let intervalInSeconds = 15;
    const COMPLETION_STATUSES = [
        PipelineExecutionStatus.FAILED,
        PipelineExecutionStatus.STOPPED,
        PipelineExecutionStatus.SUCCEEDED,
    ];

    do {
        const { PipelineExecutionStatus: status, FailureReason } =
            await sagemakerClient.send(command);

        complete = COMPLETION_STATUSES.includes(status);

        if (!complete) {
            console.log(
                `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
                again.`
            );
            await wait(intervalInSeconds);
        } else if (status === PipelineExecutionStatus.FAILED) {
```

```
        throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
        throw new Error(`Pipeline was forcefully stopped.`);
    } else {
        console.log(`Pipeline execution ${status}.`);
    }
} while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-s3').S3Client}} param0
 */
export async function getObject({ bucket, s3Client }) {
    const prefix = "output/";
    const { Contents } = await s3Client.send(
        new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
    );

    if (!Contents.length) {
        throw new Error("No objects found in bucket.");
    }

    // Find the CSV file.
    const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

    if (!outputObject) {
        throw new Error(`No CSV file found in bucket with the prefix "${prefix}.`);
    }

    const { Body } = await s3Client.send(
        new GetObjectCommand({
            Bucket: bucket,
            Key: outputObject.Key,
        }),
    );

    return Body.transformToString();
}
```

Fungsi ini adalah kutipan dari file yang menggunakan fungsi pustaka sebelumnya untuk mengatur SageMaker pipeline, menjalankannya, dan menghapus semua sumber daya yang dibuat.

```
import { retry, wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  attachPolicy,
  configureLambdaSQSEventSource,
  createLambdaExecutionPolicy,
  createLambdaExecutionRole,
  createLambdaFunction,
  createLambdaLayer,
  createS3Bucket,
  createSQSQueue,
  createSagemakerExecutionPolicy,
  createSagemakerPipeline,
  createSagemakerRole,
  getObject,
  startPipelineExecution,
  uploadCSVDataToS3,
  waitForPipelineComplete,
} from "./lib.js";
import { MESSAGES } from "./messages.js";

export class SageMakerPipelinesWkflw {
  names = {
    LAMBDA_EXECUTION_ROLE: "sagemaker-wkflw-lambda-execution-role",
    LAMBDA_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-lambda-execution-role-policy",
    LAMBDA_FUNCTION: "sagemaker-wkflw-lambda-function",
    LAMBDA_LAYER: "sagemaker-wkflw-lambda-layer",
    SAGE_MAKER_EXECUTION_ROLE: "sagemaker-wkflw-pipeline-execution-role",
    SAGE_MAKER_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-pipeline-execution-role-policy",
    SAGE_MAKER_PIPELINE: "sagemaker-wkflw-pipeline",
    SQS_QUEUE: "sagemaker-wkflw-sqs-queue",
    S3_BUCKET: `sagemaker-wkflw-s3-bucket-${Date.now()}`,
  };

  cleanUpFunctions = [];

  /**
   * @param {import("@aws-doc-sdk-examples/lib/prompter.js").Prompter} prompter
   * @param {import("@aws-doc-sdk-examples/lib/logger.js").Logger} logger
  */
}
```



```
* @param {{ IAM: import("@aws-sdk/client-iam").IAMClient, Lambda: import("@aws-
sdk/client-lambda").LambdaClient, SageMaker: import("@aws-sdk/client-
sagemaker").SageMakerClient, S3: import("@aws-sdk/client-s3").S3Client, SQS:
import("@aws-sdk/client-sqs").SQSClient }} clients
*/
constructor(prompter, logger, clients) {
  this.prompter = prompter;
  this.logger = logger;
  this.clients = clients;
}

async run() {
  try {
    await this.startWorkflow();
  } catch (err) {
    console.error(err);
    throw err;
  } finally {
    // Run all of the clean up functions. If any fail, we log the error and
continue.
    // This ensures all clean up functions are run.
    this.logger.logSeparator();
    const doCleanUp = await this.prompter.confirm({
      message: "Clean up resources?",
    });
    if (doCleanUp) {
      for (let i = this.cleanUpFunctions.length - 1; i >= 0; i--) {
        await retry(
          { intervalInMs: 1000, maxRetries: 60, swallowError: true },
          this.cleanUpFunctions[i],
        );
      }
    }
  }
}

async startWorkflow() {
  this.logger.logSeparator(MESSAGES.greetingHeader);
  await this.logger.log(MESSAGES.greeting);

  this.logger.logSeparator();
  await this.logger.log(
    MESSAGES.creatingRole.replace(
      "${ROLE_NAME}",
```

```
        this.names.LAMBDA_EXECUTION_ROLE,
    ),
);

// Create an IAM role that will be assumed by the AWS Lambda function. This
function
// is triggered by Amazon SQS messages and calls SageMaker and SageMaker
GeoSpatial actions.
const { arn: lambdaExecutionRoleArn, cleanUp: lambdaExecutionRoleCleanUp } =
    await createLambdaExecutionRole({
        name: this.names.LAMBDA_EXECUTION_ROLE,
        iamClient: this.clients.IAM,
    });
// Add a clean up step to a stack for every resource created.
this.cleanUpFunctions.push(lambdaExecutionRoleCleanUp);

await this.logger.log(
    MESSAGES.roleCreated.replace(
        "${ROLE_NAME}",
        this.names.LAMBDA_EXECUTION_ROLE,
    ),
);

this.logger.logSeparator();

await this.logger.log(
    MESSAGES.creatingRole.replace(
        "${ROLE_NAME}",
        this.names.SAGE_MAKER_EXECUTION_ROLE,
    ),
);

// Create an IAM role that will be assumed by the SageMaker pipeline. The
pipeline
// sends messages to an Amazon SQS queue and puts/retrieves Amazon S3 objects.
const {
    arn: pipelineExecutionRoleArn,
    cleanUp: pipelineExecutionRoleCleanUp,
} = await createSagemakerRole({
    iamClient: this.clients.IAM,
    name: this.names.SAGE_MAKER_EXECUTION_ROLE,
});
this.cleanUpFunctions.push(pipelineExecutionRoleCleanUp);
```

```
await this.logger.log(
  MESSAGES.roleCreated.replace(
    "${ROLE_NAME}",
    this.names.SAGE_MAKER_EXECUTION_ROLE,
  ),
);

this.logger.logSeparator();

// Create an IAM policy that allows the AWS Lambda function to invoke SageMaker APIs.
const {
  arn: lambdaExecutionPolicyArn,
  policy: lambdaPolicy,
  cleanUp: lambdaExecutionPolicyCleanUp,
} = await createLambdaExecutionPolicy({
  name: this.names.LAMBDA_EXECUTION_ROLE_POLICY,
  s3BucketName: this.names.S3_BUCKET,
  iamClient: this.clients.IAM,
  pipelineExecutionRoleArn,
});
this.cleanUpFunctions.push(lambdaExecutionPolicyCleanUp);

console.log(JSON.stringify(lambdaPolicy, null, 2), "\n");

await this.logger.log(
  MESSAGES.attachPolicy
    .replace("${POLICY_NAME}", this.names.LAMBDA_EXECUTION_ROLE_POLICY)
    .replace("${ROLE_NAME}", this.names.LAMBDA_EXECUTION_ROLE),
);

await this.prompter.checkContinue();

// Attach the Lambda execution policy to the execution role.
const { cleanUp: lambdaExecutionRolePolicyCleanUp } = await attachPolicy({
  roleName: this.names.LAMBDA_EXECUTION_ROLE,
  policyArn: lambdaExecutionPolicyArn,
  iamClient: this.clients.IAM,
});
this.cleanUpFunctions.push(lambdaExecutionRolePolicyCleanUp);

await this.logger.log(MESSAGES.policyAttached);

this.logger.logSeparator();
```

```
// Create Lambda layer for SageMaker packages.
const { versionArn: layerVersionArn, cleanUp: lambdaLayerCleanUp } =
  await createLambdaLayer({
    name: this.names.LAMBDA_LAYER,
    lambdaClient: this.clients.Lambda,
  });
this.cleanUpFunctions.push(lambdaLayerCleanUp);

await this.logger.log(
  MESSAGES.creatingFunction.replace(
    "${FUNCTION_NAME}",
    this.names.LAMBDA_FUNCTION,
  ),
);

// Create the Lambda function with the execution role.
const { arn: lambdaArn, cleanUp: lambdaCleanUp } =
  await createLambdaFunction({
    roleArn: lambdaExecutionRoleArn,
    lambdaClient: this.clients.Lambda,
    name: this.names.LAMBDA_FUNCTION,
    layerVersionArn,
  });
this.cleanUpFunctions.push(lambdaCleanUp);

await this.logger.log(
  MESSAGES.functionCreated.replace(
    "${FUNCTION_NAME}",
    this.names.LAMBDA_FUNCTION,
  ),
);

this.logger.logSeparator();

await this.logger.log(
  MESSAGES.creatingSQSQueue.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

// Create an SQS queue for the SageMaker pipeline.
const {
  queueUrl,
  queueArn,
  cleanUp: queueCleanUp,
```

```
} = await createSQSQueue({
  name: this.names.SQS_QUEUE,
  sqsClient: this.clients.SQS,
});
this.cleanupFunctions.push(queueCleanUp);

await this.logger.log(
  MESSAGES.sqsQueueCreated.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

this.logger.logSeparator();

await this.logger.log(
  MESSAGES.configuringLambdaSQSEventSource
    .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
    .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

// Configure the SQS queue as an event source for the Lambda.
const { cleanUp: lambdaSQSEventSourceCleanUp } =
  await configureLambdaSQSEventSource({
    lambdaArn,
    lambdaName: this.names.LAMBDA_FUNCTION,
    queueArn,
    sqsClient: this.clients.SQS,
    lambdaClient: this.clients.Lambda,
  });
this.cleanupFunctions.push(lambdaSQSEventSourceCleanUp);

await this.logger.log(
  MESSAGES.lambdaSQSEventSourceConfigured
    .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
    .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

this.logger.logSeparator();

// Create an IAM policy that allows the SageMaker pipeline to invoke AWS Lambda
// and send messages to the Amazon SQS queue.
const {
  arn: pipelineExecutionPolicyArn,
  policy: sagemakerPolicy,
  cleanUp: pipelineExecutionPolicyCleanUp,
} = await createSagemakerExecutionPolicy({
```

```
    sqsQueueArn: queueArn,
    lambdaArn,
    iamClient: this.clients.IAM,
    name: this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY,
    s3BucketName: this.names.S3_BUCKET,
  });
  this.cleanUpFunctions.push(pipelineExecutionPolicyCleanUp);

  console.log(JSON.stringify(sagemakerPolicy, null, 2));

  await this.logger.log(
    MESSAGES.attachPolicy
      .replace("${POLICY_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY)
      .replace("${ROLE_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE),
  );

  await this.prompter.checkContinue();

  // Attach the SageMaker execution policy to the execution role.
  const { cleanUp: pipelineExecutionRolePolicyCleanUp } = await attachPolicy({
    roleName: this.names.SAGE_MAKER_EXECUTION_ROLE,
    policyArn: pipelineExecutionPolicyArn,
    iamClient: this.clients.IAM,
  });
  this.cleanUpFunctions.push(pipelineExecutionRolePolicyCleanUp);
  // Wait for the role to be ready. If the role is used immediately,
  // the pipeline will fail.
  await wait(5);

  await this.logger.log(MESSAGES.policyAttached);

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.creatingPipeline.replace(
      "${PIPELINE_NAME}",
      this.names.SAGE_MAKER_PIPELINE,
    ),
  );

  // Create the SageMaker pipeline.
  const { cleanUp: pipelineCleanUp } = await createSagemakerPipeline({
    roleArn: pipelineExecutionRoleArn,
    functionArn: lambdaArn,
```

```
    sagemakerClient: this.clients.SageMaker,
    name: this.names.SAGE_MAKER_PIPELINE,
  });
  this.cleanUpFunctions.push(pipelineCleanUp);

  await this.logger.log(
    MESSAGES.pipelineCreated.replace(
      "${PIPELINE_NAME}",
      this.names.SAGE_MAKER_PIPELINE,
    ),
  );

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.creatingS3Bucket.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
  );

  // Create an S3 bucket for storing inputs and outputs.
  const { cleanUp: s3BucketCleanUp } = await createS3Bucket({
    name: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
  });
  this.cleanUpFunctions.push(s3BucketCleanUp);

  await this.logger.log(
    MESSAGES.s3BucketCreated.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
  );

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.uploadingInputData.replace(
      "${BUCKET_NAME}",
      this.names.S3_BUCKET,
    ),
  );

  // Upload CSV Lat/Long data to S3.
  await uploadCSVDataToS3({
    bucketName: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
  });
```

```
await this.logger.log(MESSAGES.inputDataUploaded);

this.logger.logSeparator();

await this.prompter.checkContinue(MESSAGES.executePipeline);

// Execute the SageMaker pipeline.
const { arn: pipelineExecutionArn } = await startPipelineExecution({
  name: this.names.SAGE_MAKER_PIPELINE,
  sagemakerClient: this.clients.SageMaker,
  roleArn: pipelineExecutionRoleArn,
  bucketName: this.names.S3_BUCKET,
  queueUrl,
});

// Wait for the pipeline execution to finish.
await waitForPipelineComplete({
  arn: pipelineExecutionArn,
  sagemakerClient: this.clients.SageMaker,
});

this.logger.logSeparator();

await this.logger.log(MESSAGES.outputDelay);

// The getOutput function will throw an error if the output is not
// found. The retry function will retry a failed function call once
// ever 10 seconds for 2 minutes.
const output = await retry({ intervalInMs: 10000, maxRetries: 12 }, () =>
  getObject({
    bucket: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
  })),
);

this.logger.logSeparator();
await this.logger.log(MESSAGES.outputDataRetrieved);
console.log(output.split("\n").slice(0, 6).join("\n"));
}
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .

- [CreatePipeline](#)
- [DeletePipeline](#)
- [DescribePipelineExecution](#)
- [StartPipelineExecution](#)
- [UpdatePipeline](#)

Contoh Secrets Manager menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Secrets Manager.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Dapatkan nilai rahasia

Contoh kode berikut menunjukkan cara mendapatkan nilai rahasia Secrets Manager.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import {
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";

export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: secretName,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
  //   CreatedDate: 2023-08-08T19:29:51.294Z,
  //   Name: 'binary-secret-3873048',
  //   SecretBinary: Uint8Array(11) [
  //     98, 105, 110, 97, 114,
  //     121, 32, 100, 97, 116,
  //     97
  //   ],
  //   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
  //   VersionStages: [ 'AWSCURRENT' ]
  // }

  if (response.SecretString) {
    return response.SecretString;
  }

  if (response.SecretBinary) {
    return response.SecretBinary;
  }
};
```

- Untuk detail API, lihat [GetSecretValue](#) di Referensi AWS SDK for JavaScript API.

Amazon SES contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon SES.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Buat filter tanda terima

Contoh kode berikut menunjukkan cara membuat filter tanda terima Amazon SES yang memblokir email masuk dari alamat IP atau rentang alamat IP.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import {  
  CreateReceiptFilterCommand,
```

```
    ReceiptFilterPolicy,
  } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utills/util-string.js";

const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
        address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
        addressesOptions.
      },
      /*
        The name of the IP address filter. Only ASCII letters, numbers, underscores,
        or dashes.
        Must be less than 64 characters and start and end with a letter or number.
      */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });

  try {
    return await sesClient.send(createReceiptFilterCommand);
  } catch (err) {
    console.log("Failed to create filter.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [CreateReceiptFilter](#) di Referensi AWS SDK for JavaScript API.

Buat aturan penerimaan

Contoh kode berikut menunjukkan cara membuat aturan penerimaan Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateReceiptRuleCommand, TlsPolicy } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
  });
}
```

```
});  
};  
  
const run = async () => {  
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({  
    bucketName: S3_BUCKET_NAME,  
    emailAddresses: ["email@example.com"],  
    name: RULE_NAME,  
    ruleSet: RULE_SET_NAME,  
  });  
  
  try {  
    return await sesClient.send(s3ReceiptRuleCommand);  
  } catch (err) {  
    console.log("Failed to create S3 receipt rule.", err);  
    throw err;  
  }  
};
```

- Untuk detail API, lihat [CreateReceiptRule](#) di Referensi AWS SDK for JavaScript API.

Buat set aturan tanda terima

Contoh kode berikut menunjukkan cara membuat aturan penerimaan Amazon SES yang ditetapkan untuk mengatur aturan yang diterapkan pada email masuk.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateReceiptRuleSetCommand } from "@aws-sdk/client-ses";  
import { sesClient } from "../libs/sesClient.js";  
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";  
  
const RULE_SET_NAME = getUniqueName("RuleSetName");  
  
const createCreateReceiptRuleSetCommand = (ruleSetName) => {
```

```
    return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
  };

const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- Untuk detail API, lihat [CreateReceiptRuleSet](#) di Referensi AWS SDK for JavaScript API.

Buat template email

Contoh kode berikut menunjukkan cara membuat template email Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template
     system.
     */
  });
};
```

```
    */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
          Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [CreateTemplate](#) di Referensi AWS SDK for JavaScript API.

Hapus filter tanda terima

Contoh kode berikut menunjukkan cara menghapus filter tanda terima Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).


```
import { DeleteReceiptFilterCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [DeleteReceiptFilter](#) di Referensi AWS SDK for JavaScript API.

Hapus aturan tanda terima

Contoh kode berikut menunjukkan cara menghapus aturan penerimaan Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteReceiptRuleCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";
```

```
const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleCommand = () => {
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [DeleteReceiptRule](#) di Referensi AWS SDK for JavaScript API.

Hapus satu set aturan

Contoh kode berikut menunjukkan cara menghapus kumpulan aturan Amazon SES dan semua aturan yang dikandungnya.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
```

```
const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};

const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [DeleteReceiptRuleSet](#) di Referensi AWS SDK for JavaScript API.

Hapus template email

Contoh kode berikut menunjukkan cara menghapus template email Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);
```

```
try {
  return await sesClient.send(deleteTemplateCommand);
} catch (err) {
  console.log("Failed to delete template.", err);
  return err;
}
};
```

- Untuk detail API, lihat [DeleteTemplate](#) di Referensi AWS SDK for JavaScript API.

Hapus identitas

Contoh kode berikut menunjukkan cara menghapus identitas Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
  }
};
```

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

- Untuk detail API, lihat [DeleteIdentity](#) di Referensi AWS SDK for JavaScript API.

Dapatkan template email yang ada

Contoh kode berikut menunjukkan cara mendapatkan template email Amazon SES yang ada.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (err) {
    console.log("Failed to get email template.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [GetTemplate](#) di Referensi AWS SDK for JavaScript API.

Daftar template email

Contoh kode berikut menunjukkan cara membuat daftar template email Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [ListTemplates](#) di Referensi AWS SDK for JavaScript API.

Daftar identitas

Contoh kode berikut menunjukkan cara mencantumkan identitas Amazon SES.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();


  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [ListIdentities](#) di Referensi AWS SDK for JavaScript API.

Daftar filter tanda terima

Contoh kode berikut menunjukkan cara membuat daftar filter tanda terima Amazon SES.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
};
```

- Untuk detail API, lihat [ListReceiptFilters](#) di Referensi AWS SDK for JavaScript API.

Kirim email template massal

Contoh kode berikut menunjukkan cara mengirim email template ke beberapa tujuan dengan Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
```



```

*/
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map
     each user
     * to a 'Destination' and provide user specific replacement data to create
     personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
};

```

```
);
try {
  return await sesClient.send(sendBulkTemplateEmailCommand);
} catch (err) {
  console.log("Failed to send bulk template email", err);
  return err;
}
};
```

- Untuk detail API, lihat [SendBulkTemplatedEmail](#) di Referensi AWS SDK for JavaScript API.

Kirim email

Contoh kode berikut menunjukkan cara mengirim email dengan Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
```

```
    /* required */
    Html: {
      Charset: "UTF-8",
      Data: "HTML_FORMAT_BODY",
    },
    Text: {
      Charset: "UTF-8",
      Data: "TEXT_FORMAT_BODY",
    },
  },
  Subject: {
    Charset: "UTF-8",
    Data: "EMAIL_SUBJECT",
  },
},
Source: fromAddress,
ReplyToAddresses: [
  /* more items */
],
});
};

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

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

- Untuk detail API, lihat [SendEmail](#) di Referensi AWS SDK for JavaScript API.

Kirim email mentah

Contoh kode berikut menunjukkan cara mengirim email mentah dengan Amazon SES.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan [nodemailer](#) untuk mengirim email dengan lampiran.

```
import sesClientModule from "@aws-sdk/client-ses";
/**
 * nodemailer wraps the SES SDK and calls SendRawEmail. Use this for more advanced
 * functionality like adding attachments to your email.
 *
 * https://nodemailer.com/transports/ses/
 */
import nodemailer from "nodemailer";

/**
 * @param {string} from An Amazon SES verified email address.
 * @param {*} to An Amazon SES verified email address.
 */
export const sendEmailWithAttachments = (
  from = "from@example.com",
  to = "to@example.com",
) => {
  const ses = new sesClientModule.SESClient({});
  const transporter = nodemailer.createTransport({
    SES: { ses, aws: sesClientModule },
  });

  return new Promise((resolve, reject) => {
    transporter.sendMail(
      {
        from,
        to,
        subject: "Hello World",
        text: "Greetings from Amazon SES!",
        attachments: [{ content: "Hello World!", filename: "hello.txt" }],
      },
      (err, info) => {
        if (err) {
```

```
        reject(err);
      } else {
        resolve(info);
      }
    },
  );
});
};
```

- Untuk detail API, lihat [SendRawEmail](#) di Referensi AWS SDK for JavaScript API.

Kirim email template

Contoh kode berikut menunjukkan cara mengirim email template dengan Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");
```

```

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (err) {
    console.log("Failed to send template email", err);
    return err;
  }
};

```

- Untuk detail API, lihat [SendTemplatedEmail](#) di Referensi AWS SDK for JavaScript API.

Perbarui template email

Contoh kode berikut menunjukkan cara memperbarui template email Amazon SES.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [UpdateTemplate](#) di Referensi AWS SDK for JavaScript API.

Verifikasi identitas domain

Contoh kode berikut menunjukkan cara memverifikasi identitas domain dengan Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [VerifyDomainIdentity](#) di Referensi AWS SDK for JavaScript API.

Verifikasi identitas email

Contoh kode berikut menunjukkan cara memverifikasi identitas email dengan Amazon SES.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

- Untuk detail API, lihat [VerifyEmailIdentity](#) di Referensi AWS SDK for JavaScript API.

Contoh Amazon SNS menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon SNS.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Amazon SNS

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Inisialisasi klien SNS dan daftar topik di akun Anda.

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

export const helloSns = async () => {
  // The configuration object ( `{}` ) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
  `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
  const topics = [];

  for await (const page of paginatedTopics) {
```

```
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your account.`
  );
  console.log(topics.map((t) => ` * ${t.TopicArn}`).join("\n"));
};
```

- Untuk detail API, lihat [ListTopics](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Periksa apakah nomor telepon dipilih keluar

Contoh kode berikut menunjukkan cara memeriksa apakah nomor telepon dipilih untuk tidak menerima pesan Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

```
// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

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

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CheckIfPhoneNumberIsOptedOut](#) di Referensi AWS SDK for JavaScript API.

Konfirmasikan pemilik endpoint ingin menerima pesan

Contoh kode berikut menunjukkan cara mengonfirmasi pemilik titik akhir ingin menerima pesan Amazon SNS dengan memvalidasi token yang dikirim ke titik akhir dengan tindakan Berlangganan sebelumnya.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 * that are not AWS services (HTTP/S, email) need to be
 * confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 * subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
```


```
// HTTP/S, email, or in another AWS account.
new ConfirmSubscriptionCommand({
  Token: token,
  TopicArn: topicArn,
  // If this is true, the subscriber cannot unsubscribe while unauthenticated.
  AuthenticateOnUnsubscribe: "false",
}),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
// }
return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ConfirmSubscription](#) di Referensi AWS SDK for JavaScript API.

Buat topik

Contoh kode berikut menunjukkan cara membuat topik Amazon SNS.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateTopic](#) di Referensi AWS SDK for JavaScript API.

Menghapus langganan

Contoh kode berikut menunjukkan cara menghapus langganan Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
  xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
```



```
//     httpStatusCode: 200,  
//     requestId: '0178259a-9204-507c-b620-78a7570a44c6',  
//     extendedRequestId: undefined,  
//     cfId: undefined,  
//     attempts: 1,  
//     totalRetryDelay: 0  
//   }  
// }  
return response;  
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Berhenti berlangganan](#) di Referensi AWS SDK for JavaScript API.

Hapus topik

Contoh kode berikut menunjukkan cara menghapus topik Amazon SNS dan semua langganan ke topik itu.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

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

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteTopic](#) di Referensi AWS SDK for JavaScript API.

Dapatkan properti dari suatu topik

Contoh kode berikut menunjukkan cara mendapatkan properti topik Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

```
// The AWS Region can be provided here using the `region` property. If you leave it
  blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetries":0,"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
```

```
// }  
// }  
return response;  
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetTopicAttributes](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengimpor modul SDK dan klien dan memanggil API.

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set region  
AWS.config.update({ region: "REGION" });  
  
// Create promise and SNS service object  
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })  
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })  
  .promise();  
  
// Handle promise's fulfilled/rejected states  
getTopicAttribsPromise  
  .then(function (data) {  
    console.log(data);  
  })  
  .catch(function (err) {  
    console.error(err, err.stack);  
  });
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetTopicAttributes](#) di Referensi AWS SDK for JavaScript API.

Dapatkan pengaturan untuk mengirim pesan SMS

Contoh kode berikut menunjukkan cara mendapatkan pengaturan untuk mengirim pesan SMS Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
```

```
//     extendedRequestId: undefined,  
//     cfId: undefined,  
//     attempts: 1,  
//     totalRetryDelay: 0  
//   },  
//   attributes: { DefaultSMSType: 'Transactional' }  
// }  
return response;  
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetSmSatTributes](#) di Referensi API.AWS SDK for JavaScript

Buat daftar pelanggan suatu topik

Contoh kode berikut menunjukkan cara mengambil daftar pelanggan topik Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

```
/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListSubscriptions](#) di Referensi AWS SDK for JavaScript API.

Daftar topik

Contoh kode berikut menunjukkan cara membuat daftar topik Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```


- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).

- Untuk detail API, lihat [ListTopics](#) di Referensi AWS SDK for JavaScript API.

Publikasikan pesan dengan atribut

Contoh kode berikut menunjukkan cara mempublikasikan pesan dengan atribut menggunakan Amazon SNS.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Publikasikan pesan ke topik dengan opsi grup, duplikasi, dan atribut.

```
async publishMessages() {
  const message = await this.prompter.input({
    message: MESSAGES.publishMessagePrompt,
  });

  let groupId, deduplicationId, choices;

  if (this.isFifo) {
    await this.logger.log(MESSAGES.groupIdNotice);
    groupId = await this.prompter.input({
      message: MESSAGES.groupIdPrompt,
    });
  }

  if (this.autoDedup === false) {
    await this.logger.log(MESSAGES.deduplicationIdNotice);
    deduplicationId = await this.prompter.input({
      message: MESSAGES.deduplicationIdPrompt,
    });
  }

  choices = await this.prompter.checkbox({
    message: MESSAGES.messageAttributesPrompt,
    choices: toneChoices,
  });
}
```

```
await this.snsClient.send(
  new PublishCommand({
    TopicArn: this.topicArn,
    Message: message,
    ...(groupId
      ? {
          MessageGroupId: groupId,
        }
      : {}),
    ...(deduplicationId
      ? {
          MessageDeduplicationId: deduplicationId,
        }
      : {}),
    ...(choices
      ? {
          MessageAttributes: {
            tone: {
              DataType: "String.Array",
              StringValue: JSON.stringify(choices),
            },
          },
        }
      : {}),
  })),
);

const publishAnother = await this.prompter.confirm({
  message: MESSAGES.publishAnother,
});


if (publishAnother) {
  await this.publishMessages();
}
}
```

- Untuk detail API, lihat [Publikasikan](#) di Referensi AWS SDK for JavaScript API.

Publikasikan ke topik

Contoh kode berikut menunjukkan cara mempublikasikan pesan ke topik Amazon SNS.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
plain string or an object
 *
 * if you are using the `json`
 * `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
}
```

```
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
// }
return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Publikasikan](#) di Referensi AWS SDK for JavaScript API.

Atur pengaturan default untuk mengirim pesan SMS

Contoh kode berikut menunjukkan cara mengatur pengaturan default untuk mengirim pesan SMS menggunakan Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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


/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional - (Default) Noncritical messages, such as marketing messages.
        // Transactional - Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [SetSmSatTributes](#) di Referensi API.AWS SDK for JavaScript

Tetapkan atribut topik

Contoh kode berikut menunjukkan cara mengatur atribut topik Amazon SNS.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
```


```
//     attempts: 1,  
//     totalRetryDelay: 0  
//   }  
// }  
return response;  
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [SetTopicAttributes](#) di Referensi AWS SDK for JavaScript API.

Berlangganan fungsi Lambda ke suatu topik

Contoh kode berikut menunjukkan cara berlangganan fungsi Lambda sehingga menerima pemberitahuan dari topik Amazon SNS.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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


```
* @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
* @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
*/
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Berlangganan](#) di Referensi AWS SDK for JavaScript API.

Berlangganan aplikasi seluler ke suatu topik

Contoh kode berikut menunjukkan cara berlangganan titik akhir aplikasi seluler sehingga menerima pemberitahuan dari topik Amazon SNS.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 * created
 *
 * when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
```

```
// '$metadata': {  
//   httpStatusCode: 200,  
//   requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',  
//   extendedRequestId: undefined,  
//   cfId: undefined,  
//   attempts: 1,  
//   totalRetryDelay: 0  
// },  
// SubscriptionArn: 'pending confirmation'  
// }  
return response;  
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Berlangganan](#) di Referensi AWS SDK for JavaScript API.

Berlangganan antrian SQS ke topik

Contoh kode berikut menunjukkan cara berlangganan antrian Amazon SQS sehingga menerima pemberitahuan dari topik Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";  
  
const client = new SNSClient({});  
  
export const subscribeQueue = async (  
  topicArn = "TOPIC_ARN",  
  queueArn = "QUEUE_ARN",  
) => {  
  const command = new SubscribeCommand({  
    TopicArn: topicArn,  
    Protocol: "sqs",  
  });
```

```
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
  test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

- Untuk detail API, lihat [Berlangganan](#) di Referensi AWS SDK for JavaScript API.

Berlangganan alamat email ke suatu topik

Contoh kode berikut menunjukkan cara berlangganan alamat email ke topik Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien dalam modul terpisah dan ekspor klien tersebut.

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

// The AWS Region can be provided here using the `region` property. If you leave it
blank
```

```
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Mengimpor modul SDK dan klien dan memanggil API.

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

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 * subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Berlangganan](#) di Referensi AWS SDK for JavaScript API.

Berlangganan dengan filter ke topik

Contoh kode berikut menunjukkan cara berlangganan dengan filter ke topik Amazon SNS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute set
      // to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  },
);

const response = await client.send(command);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
```

```
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
// }
return response;
};
```

- Untuk detail API, lihat [Berlangganan](#) di Referensi AWS SDK for JavaScript API.

Skenario

Publikasikan pesan ke antrian

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat topik (FIFO atau non-FIFO).
- Berlangganan beberapa antrian ke topik dengan opsi untuk menerapkan filter.
- Publikasikan pesan ke topik.
- Polling antrian untuk pesan yang diterima.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Ini adalah titik masuk untuk alur kerja ini.

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

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";
```

```
export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Kode sebelumnya menyediakan dependensi yang diperlukan dan memulai alur kerja. Bagian selanjutnya berisi sebagian besar contoh.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
string }[]}
   */
  queues = [];
  prompter;
```

```
/**
 * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
 * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
 * @param {import('../libs/prompter.js').Prompter} prompter
 * @param {import('../libs/logger.js').Logger} logger
 */
constructor(snsClient, sqsClient, prompter, logger) {
  this.snsClient = snsClient;
  this.sqsClient = sqsClient;
  this.prompter = prompter;
  this.logger = logger;
}

async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
  }
}
```



```
const response = await this.snsClient.send(
  new CreateTopicCommand({
    Name: this.topicName,
    Attributes: {
      FifoTopic: this.isFifo ? "true" : "false",
      ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
    },
  }),
);

this.topicArn = response.TopicArn;

await this.logger.log(
  MESSAGES.topicCreatedNotice
    .replace("${TOPIC_NAME}", this.topicName)
    .replace("${TOPIC_ARN}", this.topicArn),
);
}

async createQueues() {
  await this.logger.log(MESSAGES.createQueuesNotice);
  // Increase this number to add more queues.
  let maxQueues = 2;

  for (let i = 0; i < maxQueues; i++) {
    await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
    let queueName = await this.prompter.input({
      message: MESSAGES.queueNamePrompt.replace(
        "${EXAMPLE_NAME}",
        i === 0 ? "good-news" : "bad-news",
      ),
    });
  });

  if (this.isFifo) {
    queueName += ".fifo";
    await this.logger.log(MESSAGES.appendFifoNotice);
  }

  const response = await this.sqsClient.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
    }),
  );
}
```

```
);

const { Attributes } = await this.sqsClient.send(
  new GetQueueAttributesCommand({
    QueueUrl: response.QueueUrl,
    AttributeNames: ["QueueArn"],
  }),
);

this.queues.push({
  queueName,
  queueArn: Attributes.QueueArn,
  queueUrl: response.QueueUrl,
});

await this.logger.log(
  MESSAGES.queueCreatedNotice
    .replace("${QUEUE_NAME}", queueName)
    .replace("${QUEUE_URL}", response.QueueUrl)
    .replace("${QUEUE_ARN}", Attributes.QueueArn),
);
}
}

async attachQueueIamPolicies() {
  for (const [index, queue] of this.queues.entries()) {
    const policy = JSON.stringify(
      {
        Statement: [
          {
            Effect: "Allow",
            Principal: {
              Service: "sns.amazonaws.com",
            },
            Action: "sqs:SendMessage",
            Resource: queue.queueArn,
            Condition: {
              ArnEquals: {
                "aws:SourceArn": this.topicArn,
              },
            },
          },
        ],
      },
    ),
  },
);
},
},
};
```

```
        null,
        2,
    );

    if (index !== 0) {
        this.logger.logSeparator();
    }

    await this.logger.log(MESSAGES.attachPolicyNotice);
    console.log(policy);
    const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
            "${QUEUE_NAME}",
            queue.queueName,
        ),
    });

    if (addPolicy) {
        await this.sqsClient.send(
            new SetQueueAttributesCommand({
                QueueUrl: queue.queueUrl,
                Attributes: {
                    Policy: policy,
                },
            }),
        );
        queue.policy = policy;
    } else {
        await this.logger.log(
            MESSAGES.policyNotAttachedNotice.replace(
                "${QUEUE_NAME}",
                queue.queueName,
            ),
        );
    }
}

async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
        /**
         * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
         */
        const subscribeParams = {
```

```
    TopicArn: this.topicArn,
    Protocol: "sqs",
    Endpoint: queue.queueArn,
  };
  let tones = [];

  if (this.isFifo) {
    if (index === 0) {
      await this.logger.log(MESSAGES.fifoFilterNotice);
    }
    tones = await this.prompter.checkbox({
      message: MESSAGES.fifoFilterSelect.replace(
        "${QUEUE_NAME}",
        queue.queueName,
      ),
      choices: toneChoices,
    });

    if (tones.length) {
      subscribeParams.Attributes = {
        FilterPolicyScope: "MessageAttributes",
        FilterPolicy: JSON.stringify({
          tone: tones,
        }),
      };
    }
  }

  const { SubscriptionArn } = await this.snsClient.send(
    new SubscribeCommand(subscribeParams),
  );

  this.subscriptionArns.push(SubscriptionArn);

  await this.logger.log(
    MESSAGES.queueSubscribedNotice
      .replace("${QUEUE_NAME}", queue.queueName)
      .replace("${TOPIC_NAME}", this.topicName)
      .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
  );
}
}

async publishMessages() {
```

```
const message = await this.prompter.input({
  message: MESSAGES.publishMessagePrompt,
});

let groupId, deduplicationId, choices;

if (this.isFifo) {
  await this.logger.log(MESSAGES.groupIdNotice);
  groupId = await this.prompter.input({
    message: MESSAGES.groupIdPrompt,
  });

  if (this.autoDedup === false) {
    await this.logger.log(MESSAGES.deduplicationIdNotice);
    deduplicationId = await this.prompter.input({
      message: MESSAGES.deduplicationIdPrompt,
    });
  }

  choices = await this.prompter.checkbox({
    message: MESSAGES.messageAttributesPrompt,
    choices: toneChoices,
  });
}

await this.snsClient.send(
  new PublishCommand({
    TopicArn: this.topicArn,
    Message: message,
    ...(groupId
      ? {
          MessageGroupId: groupId,
        }
      : {}),
    ...(deduplicationId
      ? {
          MessageDeduplicationId: deduplicationId,
        }
      : {}),
    ...(choices
      ? {
          MessageAttributes: {
            tone: {
              DataType: "String.Array",
            }
          }
        }
      : {}),
  })
);
```

```
        StringValue: JSON.stringify(choices),
      },
    },
  : {}),
}),
);

const publishAnother = await this.prompter.confirm({
  message: MESSAGES.publishAnother,
});

if (publishAnother) {
  await this.publishMessages();
}
}

async receiveAndDeleteMessages() {
  for (const queue of this.queues) {
    const { Messages } = await this.sqsClient.send(
      new ReceiveMessageCommand({
        QueueUrl: queue.queueUrl,
      }),
    );

    if (Messages) {
      await this.logger.log(
        MESSAGES.messagesReceivedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
      console.log(Messages);

      await this.sqsClient.send(
        new DeleteMessageBatchCommand({
          QueueUrl: queue.queueUrl,
          Entries: Messages.map((message) => ({
            Id: message.MessageId,
            ReceiptHandle: message.ReceiptHandle,
          })),
        }),
      );
    } else {
```

```
        await this.logger.log(
            MESSAGES.noMessagesReceivedNotice.replace(
                "${QUEUE_NAME}",
                queue.queueName,
            ),
        );
    }
}

const deleteAndPoll = await this.prompter.confirm({
    message: MESSAGES.deleteAndPollConfirmation,
});

if (deleteAndPoll) {
    await this.receiveAndDeleteMessages();
}

async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
        await this.snsClient.send(
            new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
        );
    }

    for (const queue of this.queues) {
        await this.sqsClient.send(
            new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
        );
    }

    if (this.topicArn) {
        await this.snsClient.send(
            new DeleteTopicCommand({ TopicArn: this.topicArn }),
        );
    }
}

async start() {
    console.clear();

    try {
        this.logger.logSeparator(MESSAGES.headerWelcome);
        await this.welcome();
    }
}
```

```
    this.logger.logSeparator(MESSAGES.headerFifo);
    await this.confirmFifo();
    this.logger.logSeparator(MESSAGES.headerCreateTopic);
    await this.createTopic();
    this.logger.logSeparator(MESSAGES.headerCreateQueues);
    await this.createQueues();
    this.logger.logSeparator(MESSAGES.headerAttachPolicy);
    await this.attachQueueIamPolicies();
    this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
    await this.subscribeQueuesToTopic();
    this.logger.logSeparator(MESSAGES.headerPublishMessage);
    await this.publishMessages();
    this.logger.logSeparator(MESSAGES.headerReceiveMessages);
    await this.receiveAndDeleteMessages();
  } catch (err) {
    console.error(err);
  } finally {
    await this.destroyResources();
  }
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publikasikan](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Berlangganan](#)
 - [Berhenti berlangganan](#)

Contoh Amazon SQS menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon SQS.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Amazon SQS

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Inisialisasi klien Amazon SQS dan daftar antrian.

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object ( `{}` ) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
  `ListQueuesCommand`
```

```
// with the `NextToken` parameter from the previous request.
const paginatedQueues = paginateListQueues({ client }, {});
const queues = [];

for await (const page of paginatedQueues) {
  if (page.QueueUrls?.length) {
    queues.push(...page.QueueUrls);
  }
}

const suffix = queues.length === 1 ? "" : "s";

console.log(
  `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your account.`
);
console.log(queues.map((t) => ` * ${t}`).join("\n"));
};
```

- Untuk detail API, lihat [ListQueues](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Ubah visibilitas batas waktu pesan

Contoh kode berikut menunjukkan cara mengubah visibilitas batas waktu pesan Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Terima pesan Amazon SQS dan ubah visibilitas batas waktunya.

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 1,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- Untuk detail API, lihat [ChangeMessageVisibility](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Terima pesan Amazon SQS dan ubah visibilitas batas waktunya.

```
// 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 informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ChangeMessageVisibility](#) di Referensi AWS SDK for JavaScript API.

Konfigurasi antrian huruf mati

Contoh kode berikut menunjukkan cara mengonfigurasi antrian huruf mati di Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.

```

```
// https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-dead-letter-queues.html
    deadLetterTargetArn: deadLetterQueueArn,
    maxReceiveCount: "10",
  }},
},
QueueUrl: queueUrl,
});

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

- Untuk detail API, lihat [SetQueueAttributes](#) di Referensi AWS SDK for JavaScript API.

Membuat antrean

Contoh kode berikut menunjukkan cara membuat antrian Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat antrean standar Amazon SQS.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
```

```
    MessageRetentionPeriod: "86400",
  },
});

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

Buat antrian Amazon SQS dengan polling panjang.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: {
        // When the wait time for the ReceiveMessage API action is greater than 0,
        // long polling is in effect. The maximum long polling wait time is 20
        // seconds. Long polling helps reduce the cost of using Amazon SQS by,
        // eliminating the number of empty responses and false empty responses.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
        SQSDeveloperGuide/sqs-short-and-long-polling.html
        ReceiveMessageWaitTimeSeconds: "20",
      },
    }),
  );
  console.log(response);
  return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateQueue](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat antrian standar Amazon SQS.

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

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

Buat antrian Amazon SQS yang menunggu pesan tiba.

```
// 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 informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateQueue](#) di Referensi AWS SDK for JavaScript API.

Menghapus sekumpulan pesan dari antrian

Contoh kode berikut menunjukkan cara menghapus sekumpulan pesan dari antrian Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";
```

```
const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Untuk detail API, lihat [DeleteMessageBatch](#) di Referensi AWS SDK for JavaScript API.

Menghapus pesan dari antrian

Contoh kode berikut menunjukkan cara menghapus pesan dari antrian Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menerima dan menghapus pesan Amazon SQS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    })
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
```

```
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Untuk detail API, lihat [DeleteMessage](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menerima dan menghapus pesan Amazon SQS.

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

// Create an SQS service object
```

```
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteMessage](#) di Referensi AWS SDK for JavaScript API.

Hapus antrian

Contoh kode berikut menunjukkan cara menghapus antrian Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus antrian Amazon SQS.

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteQueue](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus antrian Amazon SQS.

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

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

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteQueue](#) di Referensi AWS SDK for JavaScript API.

Dapatkan atribut untuk antrian

Contoh kode berikut menunjukkan cara mendapatkan atribut untuk antrian Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
```

```
    AttributeNames: ["DelaySeconds"],
  });


  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- Untuk detail API, lihat [GetQueueAttributes](#) di Referensi AWS SDK for JavaScript API.

Dapatkan URL antrian

Contoh kode berikut menunjukkan cara mendapatkan URL antrian Amazon SQS.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan URL untuk antrean Amazon SQS.

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";
```



```
export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetQueueUrl](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan URL untuk antrian Amazon SQS.

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

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetQueueUrl](#) di Referensi AWS SDK for JavaScript API.

Daftar antrian

Contoh kode berikut menunjukkan cara membuat daftar antrian Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar antrian Amazon SQS Anda.

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    urls.forEach((url) => console.log(url));
  }

  return urls;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListQueues](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar antrian Amazon SQS Anda.

```
// 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 informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListQueues](#) di Referensi AWS SDK for JavaScript API.

Menerima pesan dari antrian

Contoh kode berikut menunjukkan cara menerima pesan dari antrian Amazon SQS.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menerima pesan dari antrian Amazon SQS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    })
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
```

```
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
    })),
    );
} else {
    await client.send(
        new DeleteMessageBatchCommand({
            QueueUrl: queueUrl,
            Entries: Messages.map((message) => ({
                Id: message.MessageId,
                ReceiptHandle: message.ReceiptHandle,
            })),
        })),
    );
}
};
```

Menerima pesan dari antrian Amazon SQS menggunakan dukungan polling panjang.

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
    const command = new ReceiveMessageCommand({
        AttributeNames: ["SentTimestamp"],
        MaxNumberOfMessages: 1,
        MessageAttributeNames: ["All"],
        QueueUrl: queueUrl,
        // The duration (in seconds) for which the call waits for a message
        // to arrive in the queue before returning. If a message is available,
        // the call returns sooner than WaitTimeSeconds. If no messages are
        // available and the wait time expires, the call returns successfully
        // with an empty list of messages.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
        API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
        WaitTimeSeconds: 20,
    });

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

```
    return response;
};
```

- Untuk detail API, lihat [ReceiveMessage](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menerima pesan dari antrian Amazon SQS menggunakan dukungan polling panjang.

```
// 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 informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ReceiveMessage](#) di Referensi AWS SDK for JavaScript API.

Mengirim Pesan Antrian

Contoh kode berikut menunjukkan cara mengirim pesan ke antrian Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kirim pesan ke antrian Amazon SQS.

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
  },
  },
  MessageBody:
```

```
    "Information about current NY Times fiction bestseller for week of
    12/11/2016.",
  });

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [SendMessage](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kirim pesan ke antrian Amazon SQS.

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

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
  },
};
```



```
WeeksOn: {
  DataType: "Number",
  StringValue: "6",
},
},
MessageBody:
  "Information about current NY Times fiction bestseller for week of 12/11/2016.",
// MessageDeduplicationId: "TheWhistler", // Required for FIFO queues
// MessageGroupId: "Group1", // Required for FIFO queues
QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [SendMessage](#) di Referensi AWS SDK for JavaScript API.

Atur atribut antrian

Contoh kode berikut menunjukkan cara mengatur atribut untuk antrian Amazon SQS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";
```

```
export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

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

Konfigurasi antrian Amazon SQS untuk menggunakan polling panjang.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
  });

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

- Untuk detail API, lihat [SetQueueAttributes](#) di Referensi AWS SDK for JavaScript API.

Skenario

Publikasikan pesan ke antrian

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat topik (FIFO atau non-FIFO).
- Berlangganan beberapa antrian ke topik dengan opsi untuk menerapkan filter.
- Publikasikan pesan ke topik.
- Polling antrian untuk pesan yang diterima.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Ini adalah titik masuk untuk alur kerja ini.

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

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Kode sebelumnya menyediakan dependensi yang diperlukan dan memulai alur kerja. Bagian selanjutnya berisi sebagian besar contoh.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../libs/prompter.js').Prompter} prompter
   * @param {import('../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
```

```
await this.logger.log(MESSAGES.snsFifoDescription);
this.isFifo = await this.prompter.confirm({
  message: MESSAGES.snsFifoPrompt,
});

if (this.isFifo) {
  this.logger.logSeparator(MESSAGES.headerDedup);
  await this.logger.log(MESSAGES.deduplicationNotice);
  await this.logger.log(MESSAGES.deduplicationDescription);
  this.autoDedup = await this.prompter.confirm({
    message: MESSAGES.deduplicationPrompt,
  });
}
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
  }

  const response = await this.snsClient.send(
    new CreateTopicCommand({
      Name: this.topicName,
      Attributes: {
        FifoTopic: this.isFifo ? "true" : "false",
        ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
      },
    }),
  );

  this.topicArn = response.TopicArn;

  await this.logger.log(
    MESSAGES.topicCreatedNotice
      .replace("${TOPIC_NAME}", this.topicName)
      .replace("${TOPIC_ARN}", this.topicArn),
  );
}
```

```
async createQueues() {
  await this.logger.log(MESSAGES.createQueuesNotice);
  // Increase this number to add more queues.
  let maxQueues = 2;

  for (let i = 0; i < maxQueues; i++) {
    await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
    let queueName = await this.prompter.input({
      message: MESSAGES.queueNamePrompt.replace(
        "${EXAMPLE_NAME}",
        i === 0 ? "good-news" : "bad-news",
      ),
    });

    if (this.isFifo) {
      queueName += ".fifo";
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.sqsClient.send(
      new CreateQueueCommand({
        QueueName: queueName,
        Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
      }),
    );

    const { Attributes } = await this.sqsClient.send(
      new GetQueueAttributesCommand({
        QueueUrl: response.QueueUrl,
        AttributeNames: ["QueueArn"],
      }),
    );

    this.queues.push({
      queueName,
      queueArn: Attributes.QueueArn,
      queueUrl: response.QueueUrl,
    });

    await this.logger.log(
      MESSAGES.queueCreatedNotice
        .replace("${QUEUE_NAME}", queueName)
        .replace("${QUEUE_URL}", response.QueueUrl)
    );
  }
}
```

```
        .replace("${QUEUE_ARN}", Attributes.QueueArn),
    );
}
}

async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
        const policy = JSON.stringify(
            {
                Statement: [
                    {
                        Effect: "Allow",
                        Principal: {
                            Service: "sns.amazonaws.com",
                        },
                        Action: "sqs:SendMessage",
                        Resource: queue.queueArn,
                        Condition: {
                            ArnEquals: {
                                "aws:SourceArn": this.topicArn,
                            },
                        },
                    },
                ],
            },
            null,
            2,
        );

        if (index !== 0) {
            this.logger.logSeparator();
        }

        await this.logger.log(MESSAGES.attachPolicyNotice);
        console.log(policy);
        const addPolicy = await this.prompter.confirm({
            message: MESSAGES.addPolicyConfirmation.replace(
                "${QUEUE_NAME}",
                queue.queueName,
            ),
        });
    });

    if (addPolicy) {
        await this.sqsClient.send(
```

```
        new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
                Policy: policy,
            },
        }),
    );
    queue.policy = policy;
} else {
    await this.logger.log(
        MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
        ),
    );
}
}
}

async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
        /**
         * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
         */
        const subscribeParams = {
            TopicArn: this.topicArn,
            Protocol: "sqs",
            Endpoint: queue.queueArn,
        };
        let tones = [];

        if (this.isFifo) {
            if (index === 0) {
                await this.logger.log(MESSAGES.fifoFilterNotice);
            }
            tones = await this.prompter.checkbox({
                message: MESSAGES.fifoFilterSelect.replace(
                    "${QUEUE_NAME}",
                    queue.queueName,
                ),
                choices: toneChoices,
            });
        }

        if (tones.length) {
```



```
        subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
                tone: tones,
            }),
        };
    }
}

const { SubscriptionArn } = await this.snsClient.send(
    new SubscribeCommand(subscribeParams),
);

this.subscriptionArns.push(SubscriptionArn);

await this.logger.log(
    MESSAGES.queueSubscribedNotice
        .replace("${QUEUE_NAME}", queue.queueName)
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
);
}
}

async publishMessages() {
    const message = await this.prompter.input({
        message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
        await this.logger.log(MESSAGES.groupIdNotice);
        groupId = await this.prompter.input({
            message: MESSAGES.groupIdPrompt,
        });
    }

    if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
            message: MESSAGES.deduplicationIdPrompt,
        });
    }
}
```

```
    choices = await this.prompter.checkbox({
      message: MESSAGES.messageAttributesPrompt,
      choices: toneChoices,
    });
  }

  await this.snsClient.send(
    new PublishCommand({
      TopicArn: this.topicArn,
      Message: message,
      ...(groupId
        ? {
            MessageGroupId: groupId,
          }
        : {}),
      ...(deduplicationId
        ? {
            MessageDeduplicationId: deduplicationId,
          }
        : {}),
      ...(choices
        ? {
            MessageAttributes: {
              tone: {
                DataType: "String.Array",
                StringValue: JSON.stringify(choices),
              },
            },
          }
        : {}),
    })),
  );

  const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
  });

  if (publishAnother) {
    await this.publishMessages();
  }
}

async receiveAndDeleteMessages() {
  for (const queue of this.queues) {
```

```
const { Messages } = await this.sqsClient.send(
  new ReceiveMessageCommand({
    QueueUrl: queue.queueUrl,
  }),
);

if (Messages) {
  await this.logger.log(
    MESSAGES.messagesReceivedNotice.replace(
      "${QUEUE_NAME}",
      queue.queueName,
    ),
  );
  console.log(Messages);

  await this.sqsClient.send(
    new DeleteMessageBatchCommand({
      QueueUrl: queue.queueUrl,
      Entries: Messages.map((message) => ({
        Id: message.MessageId,
        ReceiptHandle: message.ReceiptHandle,
      })),
    }),
  );
} else {
  await this.logger.log(
    MESSAGES.noMessagesReceivedNotice.replace(
      "${QUEUE_NAME}",
      queue.queueName,
    ),
  );
}

const deleteAndPoll = await this.prompter.confirm({
  message: MESSAGES.deleteAndPollConfirmation,
});

if (deleteAndPoll) {
  await this.receiveAndDeleteMessages();
}

async destroyResources() {
```

```
for (const subscriptionArn of this.subscriptionArns) {
  await this.snsClient.send(
    new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
  );
}

for (const queue of this.queues) {
  await this.sqsClient.send(
    new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
  );
}

if (this.topicArn) {
  await this.snsClient.send(
    new DeleteTopicCommand({ TopicArn: this.topicArn }),
  );
}
}

async start() {
  console.clear();

  try {
    this.logger.logSeparator(MESSAGES.headerWelcome);
    await this.welcome();
    this.logger.logSeparator(MESSAGES.headerFifo);
    await this.confirmFifo();
    this.logger.logSeparator(MESSAGES.headerCreateTopic);
    await this.createTopic();
    this.logger.logSeparator(MESSAGES.headerCreateQueues);
    await this.createQueues();
    this.logger.logSeparator(MESSAGES.headerAttachPolicy);
    await this.attachQueueIamPolicies();
    this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
    await this.subscribeQueuesToTopic();
    this.logger.logSeparator(MESSAGES.headerPublishMessage);
    await this.publishMessages();
    this.logger.logSeparator(MESSAGES.headerReceiveMessages);
    await this.receiveAndDeleteMessages();
  } catch (err) {
    console.error(err);
  } finally {
    await this.destroyResources();
  }
}
```

```
}  
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publikasikan](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Berlangganan](#)
 - [Berhenti berlangganan](#)

Contoh Step Functions menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Step Functions.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Mulai menjalankan mesin status

Contoh kode berikut menunjukkan bagaimana memulai Step Functions state machine run.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
}
```

```
// }
return response;
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfncClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- Untuk detail API, lihat [StartExecution](#) di Referensi AWS SDK for JavaScript API.

AWS STS contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) with AWS STS.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Asumsikan peran

Contoh kode berikut menunjukkan bagaimana untuk mengambil peran dengan AWS STS.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Asumsikan peran IAM.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
}
```



```
}  
};
```

- Untuk detail API, lihat [AssumeRole](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// Load the AWS SDK for Node.js  
const AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
var roleToAssume = {  
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",  
  RoleSessionName: "session1",  
  DurationSeconds: 900,  
};  
var roleCreds;  
  
// Create the STS service object  
var sts = new AWS.STS({ apiVersion: "2011-06-15" });  
  
//Assume Role  
sts.assumeRole(roleToAssume, function (err, data) {  
  if (err) console.log(err, err.stack);  
  else {  
    roleCreds = {  
      accessKeyId: data.Credentials.AccessKeyId,  
      secretAccessKey: data.Credentials.SecretAccessKey,  
      sessionToken: data.Credentials.SessionToken,  
    };  
    sts.getCallerIdentity(roleCreds);  
  }  
});  
  
//Get Arn of current identity
```

```
function stsGetCallerIdentity(creds) {
  var stsParams = { credentials: creds };
  // Create STS service object
  var sts = new AWS.STS(stsParams);

  sts.getCallerIdentity({}, function (err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data.Arn);
    }
  });
}
```

- Untuk detail API, lihat [AssumeRole](#) di Referensi AWS SDK for JavaScript API.

AWS Support contoh menggunakan SDK untuk JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) with AWS Support.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.


Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo AWS Support

Contoh kode berikut menunjukkan cara untuk mulai menggunakan AWS Support.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Panggil `main ()` untuk menjalankan contoh.

```
import {
  DescribeServicesCommand,
  SupportClient,
} from "@aws-sdk/client-support";

// Change the value of 'region' to your preferred AWS Region.
const client = new SupportClient({ region: "us-east-1" });

const getServiceCount = async () => {
  try {
    const { services } = await client.send(new DescribeServicesCommand({}));
    return services.length;
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
    } else {
      throw err;
    }
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- Untuk detail API, lihat [DescribeServices](#) di Referensi AWS SDK for JavaScript API.

Topik

- [Tindakan](#)
- [Skenario](#)

Tindakan

Tambahkan komunikasi ke kasus

Contoh kode berikut menunjukkan cara menambahkan AWS Support komunikasi dengan lampiran ke kasus dukungan.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { AddCommunicationToCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  let attachmentSetId;

  try {
    // Add a communication to a case.
    const response = await client.send(
      new AddCommunicationToCaseCommand({
        communicationBody: "Adding an attachment.",
        // Set value to an existing support case id.
        caseId: "CASE_ID",
        // Optional. Set value to an existing attachment set id to add attachments
        // to the case.
        attachmentSetId,
      }),
    );
  }
```


```
    console.log(response);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [AddCommunicationToCase](#) di Referensi AWS SDK for JavaScript API.

Tambahkan lampiran ke satu set

Contoh kode berikut menunjukkan cara menambahkan AWS Support lampiran ke set lampiran.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { AddAttachmentsToSetCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new attachment set or add attachments to an existing set.
    // Provide an 'attachmentSetId' value to add attachments to an existing set.
    // Use AddCommunicationToCase or CreateCase to associate an attachment set with
    a support case.
    const response = await client.send(
      new AddAttachmentsToSetCommand({
        // You can add up to three attachments per set. The size limit is 5 MB per
        attachment:
        attachments: [
          {
            fileName: "example.txt",
            data: new TextEncoder().encode("some example text"),
          },
        ],
      })
    );
  }
};
```

```
    }),
  );
  // Use this ID in AddCommunicationToCase or CreateCase.
  console.log(response.attachmentSetId);
  return response;
} catch (err) {
  console.error(err);
}
};
```

- Untuk detail API, lihat [AddAttachmentsToSet](#) di Referensi AWS SDK for JavaScript API.

Buat kasus

Contoh kode berikut menunjukkan cara membuat AWS Support kasus baru.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { CreateCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new case and log the case id.
    // Important: This creates a real support case in your account.
    const response = await client.send(
      new CreateCaseCommand({
        // The subject line of the case.
        subject: "IGNORE: Test case",
        // Use DescribeServices to find available service codes for each service.
        serviceCode: "service-quicksight-end-user",
        // Use DescribeSecurityLevels to find available severity codes for your
        support plan.
        severityCode: "low",
```

```
    // Use DescribeServices to find available category codes for each service.
    categoryCode: "end-user-support",
    // The main description of the support case.
    communicationBody: "This is a test. Please ignore.",
  )),
);
console.log(response.caseId);
return response;
} catch (err) {
  console.error(err);
}
};
```

- Untuk detail API, lihat [CreateCase](#) di Referensi AWS SDK for JavaScript API.

Jelaskan lampiran

Contoh kode berikut menunjukkan bagaimana mendeskripsikan lampiran untuk sebuah AWS Support kasus.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeAttachmentCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the metadata and content of an attachment.
    const response = await client.send(
      new DescribeAttachmentCommand({
        // Set value to an existing attachment id.
        // Use DescribeCommunications or DescribeCases to find an attachment id.
        attachmentId: "ATTACHMENT_ID",
      }),
    ),
```

```
);  
console.log(response.attachment?.fileName);  
return response;  
} catch (err) {  
  console.error(err);  
}  
};
```

- Untuk detail API, lihat [DescribeAttachment](#) di Referensi AWS SDK for JavaScript API.

Jelaskan kasus

Contoh kode berikut menunjukkan bagaimana menggambarkan AWS Support kasus.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeCasesCommand } from "@aws-sdk/client-support";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  try {  
    // Get all of the unresolved cases in your account.  
    // Filter or expand results by providing parameters to the DescribeCasesCommand.  
    Refer  
    // to the TypeScript definition and the API doc for more information on possible  
    parameters.  
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-  
support/interfaces/describecasescommandinput.html  
    const response = await client.send(new DescribeCasesCommand({}));  
    const caseIds = response.cases.map((supportCase) => supportCase.caseId);  
    console.log(caseIds);  
    return response;  
  } catch (err) {  
    console.error(err);  
  }  
};
```




```
}  
};
```

- Untuk detail API, lihat [DescribeCases](#) di Referensi AWS SDK for JavaScript API.

Jelaskan komunikasi

Contoh kode berikut menunjukkan bagaimana mendeskripsikan AWS Support komunikasi untuk suatu kasus.

SDK untuk JavaScript (v3)

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeCommunicationsCommand } from "@aws-sdk/client-support";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  try {  
    // Get all communications for the support case.  
    // Filter results by providing parameters to the DescribeCommunicationsCommand.  
    Refer  
    // to the TypeScript definition and the API doc for more information on possible  
    parameters.  
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-  
support/interfaces/describecommunicationscommandinput.html  
    const response = await client.send(  
      new DescribeCommunicationsCommand({  
        // Set value to an existing case id.  
        caseId: "CASE_ID",  
      }),  
    );  
    const text = response.communications.map((item) => item.body).join("\n");  
    console.log(text);  
    return response;  
  } catch (err) {
```

```
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DescribeCommunications](#) di Referensi AWS SDK for JavaScript API.

Jelaskan tingkat keparahan

Contoh kode berikut menunjukkan bagaimana menggambarkan AWS Support tingkat keparahan.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { DescribeSeverityLevelsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the list of severity levels.
    // The available values depend on the support plan for the account.
    const response = await client.send(new DescribeSeverityLevelsCommand({}));
    console.log(response.severityLevels);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [DescribeSeverityLevels](#) di Referensi AWS SDK for JavaScript API.

Selesaikan kasus

Contoh kode berikut menunjukkan cara menyelesaikan AWS Support kasus.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { ResolveCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

const main = async () => {
  try {
    const response = await client.send(
      new ResolveCaseCommand({
        caseId: "CASE_ID",
      }),
    );

    console.log(response.finalCaseStatus);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Untuk detail API, lihat [ResolveCase](#) di Referensi AWS SDK for JavaScript API.

Skenario

Memulai kasus

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Dapatkan dan tampilkan layanan yang tersedia dan tingkat keparahan untuk kasus.
- Buat kasus dukungan menggunakan layanan, kategori, dan tingkat keparahan yang dipilih.
- Dapatkan dan tampilkan daftar kasus terbuka untuk hari ini.
- Tambahkan set lampiran dan komunikasi ke kasus baru.

- Jelaskan keterikatan dan komunikasi baru untuk kasus ini.
- Selesaikan kasusnya.
- Dapatkan dan tampilkan daftar kasus yang diselesaikan untuk hari ini.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan skenario interaktif di terminal.

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import inquirer from "inquirer";

// Retry an asynchronous function on failure.
const retry = async ({ intervalInMs = 500, maxRetries = 10 }, fn) => {
  try {
    return await fn();
  } catch (err) {
    console.log(`Function call failed. Retrying.`);
    console.error(err.message);
    if (maxRetries === 0) throw err;
    await new Promise((resolve) => setTimeout(resolve, intervalInMs));
    return retry({ intervalInMs, maxRetries: maxRetries - 1 }, fn);
  }
};
```

```
const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n  ${text}\n${rule}\n`;
};

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

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});

  try {
    await client.send(command);
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature."
      );
    } else {
      throw err;
    }
  }
};

// Get the list of available services.
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const { selectedService } = await inquirer.prompt({
    name: "selectedService",
    type: "list",
    message:
      "Select a service. Your support case will be created for this service. The list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
};

// Get the list of available support case categories for a service.
export const getCategory = async (service) => {
  const { selectedCategory } = await inquirer.prompt({
    name: "selectedCategory",
    type: "list",
    message: "Select a category.",
  });
};
```

```
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const { selectedSeverityLevel } = await inquirer.prompt({
    name: "selectedSeverityLevel",
    type: "list",
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

// Create a new support case and return the caseId.
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodaysOpenCases = async () => {
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });
};
```

```
const { cases } = await client.send(command);

if (cases.length === 0) {
  throw new Error(
    "Unexpected number of cases. Expected more than 0 open cases."
  );
}
return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
  const command = new AddAttachmentsToSetCommand({
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
  });
  const { communications } = await client.send(command);
  return communications;
};

// Get an attachment set.
export const getFirstAttachment = (communications) => {
```

```
const firstCommWithAttachment = communications.find(
  (c) => c.attachmentSet.length > 0
);
return firstCommWithAttachment?.attachmentSet[0].attachmentId;
};

// Get an attachment.
export const getAttachment = async (attachmentId) => {
  const command = new DescribeAttachmentCommand({
    attachmentId,
  });
  const { attachment } = await client.send(command);
  return attachment;
};

// Resolve the case matching the given case ID.
export const resolveCase = async (caseId) => {
  const { shouldResolve } = await inquirer.prompt({
    name: "shouldResolve",
    type: "confirm",
    message: `Do you want to resolve ${caseId}?`,
  });

  if (shouldResolve) {
    const command = new ResolveCaseCommand({
      caseId: caseId,
    });

    await client.send(command);
    return true;
  }
  return false;
};

// Find a specific case in the list of provided cases by case ID.
// If the case is not found, and the results are paginated, continue
// paging through the results.
export const findCase = async ({ caseId, cases, nextToken }) => {
  const foundCase = cases.find((c) => c.caseId === caseId);

  if (foundCase) {
    return foundCase;
  }
}
```



```
if (nextToken) {
  const response = await client.send(
    new DescribeCasesCommand({
      nextToken,
      includeResolvedCases: true,
    })
  );
  return findCase({
    caseId,
    cases: response.cases,
    nextToken: response.nextToken,
  });
}

throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfDay = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfDay.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
    select one.
  }
}
```

```
const selectedCategory = await getCategory(selectedService);

// Provide the severity available severity levels for the account and prompt the
user to select one.
const selectedSeverityLevel = await getSeverityLevel();

// Create a support case.
console.log("\nCreating a support case.");
caseId = await createCase({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
});
console.log(`Support case created: ${caseId}`);

// Display a list of open support cases created today.
const todaysOpenCases = await retry(
  { intervalInMs: 1000, maxRetries: 15 },
  getTodaysOpenCases
);
console.log(
  `\nOpen support cases created today: ${todaysOpenCases.length}`
);
console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

// Create an attachment set.
console.log("\nCreating an attachment set.");
const attachmentSetId = await createAttachmentSet();
console.log(`Attachment set created: ${attachmentSetId}`);

// Add the attachment set to the support case.
console.log(`\nAdding attachment set to ${caseId}`);
await linkAttachmentSetToCase(attachmentSetId, caseId);
console.log(`Attachment set added to ${caseId}`);

// List the communications for a support case.
console.log(`\nListing communications for ${caseId}`);
const communications = await getCommunications(caseId);
console.log(
  communications
    .map(
      (c) =>
        `Communication created on ${c.timeCreated}. Has
        ${c.attachmentSet.length} attachments.`
    )
);
```

```
    )
    .join("\n")
  );

  // Describe the first attachment.
  console.log(`\nDescribing attachment ${attachmentSetId}`);
  const attachmentId = getFirstAttachment(communications);
  const attachment = await getAttachment(attachmentId);
  console.log(
    `Attachment is the file '${
      attachment.fileName
    }' with data: \n${new TextDecoder().decode(attachment.data)}`
  );

  // Confirm that the support case should be resolved.
  const isResolved = await resolveCase(caseId);
  if (isResolved) {
    // List the resolved cases and include the one previously created.
    // Resolved cases can take a while to appear.
    console.log(
      "\nWaiting for case status to be marked as resolved. This can take some
time."
    );
    const resolvedCases = await retry(
      { intervalInMs: 20000, maxRetries: 15 },
      () => getTodayResolvedCases(caseId)
    );
    console.log("Resolved cases:");
    console.log(resolvedCases.map((c) => c.caseId).join("\n"));
  }
} catch (err) {
  console.error(err);
}
};
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [AddAttachmentsToSet](#)
 - [AddCommunicationToCase](#)
 - [CreateCase](#)
 - [DescribeAttachment](#)

- [DescribeCases](#)
- [DescribeCommunications](#)
- [DescribeServices](#)
- [DescribeSeverityLevels](#)
- [ResolveCase](#)

Contoh Amazon Transcribe menggunakan SDK for JavaScript (v3)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for JavaScript (v3) dengan Amazon Transcribe.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

Tindakan

Hapus pekerjaan transkripsi medis

Contoh kode berikut menunjukkan cara menghapus pekerjaan transkripsi Amazon Transcribe Medical.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Hapus pekerjaan transkripsi medis.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
  'medical_transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteMedicalTranscriptionJob](#) di Referensi AWS SDK for JavaScript API.

Hapus pekerjaan transkripsi

Contoh kode berikut menunjukkan cara menghapus pekerjaan transkripsi Amazon Transcribe.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus pekerjaan transkripsi.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Buat klien.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
```

```
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteTranscriptionJob](#) di Referensi AWS SDK for JavaScript API.

Daftar pekerjaan transkripsi medis

Contoh kode berikut menunjukkan cara membuat daftar pekerjaan transkripsi Amazon Transcribe Medical.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Daftar pekerjaan transkripsi medis.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
```

```
OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
Media: {
  MediaFileUri: "SOURCE_FILE_LOCATION",
  // The S3 object location of the input media file. The URI must be in the same
  region
  // as the API endpoint that you are calling. For example,
  // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
},
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListMedicalTranscriptionJobs](#) di Referensi AWS SDK for JavaScript API.

Daftar pekerjaan transkripsi

Contoh kode berikut menunjukkan cara membuat daftar pekerjaan transkripsi Amazon Transcribe.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Daftar pekerjaan transkripsi.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Buat klien.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListTranscriptionJobs](#) di Referensi AWS SDK for JavaScript API.

Mulai pekerjaan transkripsi medis

Contoh kode berikut menunjukkan cara memulai pekerjaan transkripsi Amazon Transcribe Medical.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat klien.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Mulai pekerjaan transkripsi medis.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
    region
    // as the API endpoint that you are calling. For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
```

```
    },
  };

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [StartMedicalTranscriptionJob](#) di Referensi AWS SDK for JavaScript API.

Mulai pekerjaan transkripsi

Contoh kode berikut menunjukkan cara memulai pekerjaan transkripsi Amazon Transcribe.

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mulai pekerjaan transkripsi.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
```

```
LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
Media: {
  MediaFileUri: "SOURCE_LOCATION",
  // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/
hello_world.wav"
},
OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Buat klien.

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [StartTranscriptionJob](#) di Referensi AWS SDK for JavaScript API.

Contoh lintas layanan menggunakan SDK untuk JavaScript (v3)

Contoh aplikasi berikut menggunakan AWS SDK for JavaScript (v3) untuk bekerja di beberapa Layanan AWS.

Contoh lintas layanan menargetkan pengalaman tingkat lanjut untuk membantu Anda mulai membangun aplikasi.

Contoh-contoh

- [Membangun aplikasi Amazon Transcribe](#)
- [Membangun aplikasi streaming Amazon Transcribe](#)
- [Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB](#)
- [Buat chatbot Amazon Lex untuk melibatkan pengunjung situs web Anda](#)
- [Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label](#)
- [Membuat aplikasi web untuk melacak data DynamoDB](#)
- [Buat pelacak butir kerja Aurora Serverless](#)
- [Membuat aplikasi penjelajah Amazon Textract](#)
- [Buat aplikasi yang menganalisis umpan balik pelanggan dan mensintesis audio](#)
- [Mendeteksi APD dalam gambar dengan Amazon AWS Rekognition menggunakan SDK](#)
- [Mendeteksi objek dalam gambar dengan Amazon Rekognition menggunakan SDK AWS](#)
- [Mendeteksi orang dan objek dalam video dengan Amazon Rekognition menggunakan SDK AWS](#)
- [Menginvokasi fungsi Lambda dari browser](#)
- [Menggunakan API Gateway untuk menginvokasi fungsi Lambda](#)
- [Menggunakan Step Functions untuk menginvokasi fungsi Lambda](#)
- [Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda](#)

Membangun aplikasi Amazon Transcribe

SDK untuk JavaScript (v3)

Buat aplikasi yang menggunakan Amazon Transcribe untuk menyalin dan menampilkan rekaman suara di peramban. Aplikasi ini menggunakan dua bucket Amazon Simple Storage Service (Amazon S3), satu untuk meng-host kode aplikasi, dan satu lagi untuk menyimpan transkripsi. Aplikasi ini menggunakan kolam pengguna Amazon Cognito untuk mengautentikasi pengguna Anda. Pengguna yang diautentikasi memiliki izin AWS Identity and Access Management (IAM) untuk mengakses layanan yang diperlukan. AWS

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- Identitas Amazon Cognito
- Amazon S3
- Amazon Transcribe

Membangun aplikasi streaming Amazon Transcribe

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan Amazon Transcribe untuk membuat aplikasi yang merekam, menyalin, dan menerjemahkan audio langsung secara real-time, dan mengirim email hasilnya menggunakan Amazon Simple Email Service (Amazon SES).

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB

SDK untuk JavaScript (v3)

Contoh ini menunjukkan cara membangun aplikasi yang memungkinkan pengguna mengirimkan data ke tabel Amazon DynamoDB, dan mengirim pesan teks ke administrator menggunakan Amazon Simple Notification Service (Amazon SNS).

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SNS

Buat chatbot Amazon Lex untuk melibatkan pengunjung situs web Anda

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan Amazon Lex API untuk membuat Chatbot dalam aplikasi web untuk melibatkan pengunjung situs web Anda.

Untuk kode sumber lengkap dan petunjuk tentang cara mengatur dan menjalankan, lihat contoh lengkap [Membangun chatbot Amazon Lex](#) di panduan AWS SDK for JavaScript pengembang.

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label

SDK untuk JavaScript (v3)

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat posting di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Membuat aplikasi web untuk melacak data DynamoDB

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan Amazon DynamoDB API untuk membuat aplikasi web dinamis yang melacak data kerja DynamoDB.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

Buat pelacak butir kerja Aurora Serverless

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan AWS SDK for JavaScript (v3) untuk membuat aplikasi web yang melacak item pekerjaan dalam database Amazon Aurora dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan sisi depan yang dibangun dengan React.js untuk berinteraksi dengan sisi belakang Express Node.js.

- Integrasikan aplikasi web React.js dengan Layanan AWS.
- Cantumkan, tambahkan, dan perbarui butir di tabel Aurora.
- Kirim laporan email tentang butir kerja terfilter dengan menggunakan Amazon SES.
- Menyebarkan dan mengelola sumber daya contoh dengan AWS CloudFormation skrip yang disertakan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

Membuat aplikasi penjelajah Amazon Textract

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan aplikasi AWS SDK for JavaScript untuk membangun aplikasi React yang menggunakan Amazon Textract untuk mengekstrak data dari gambar dokumen dan menampilkannya di halaman web interaktif. Contoh ini berjalan di peramban web dan memerlukan identitas Amazon Cognito yang diautentikasi sebagai kredensialnya. Contoh ini menggunakan Amazon Simple Storage Service (Amazon S3) untuk penyimpanan, dan untuk notifikasi, contoh ini mengambil polling antrean Amazon Simple Queue Service (Amazon SQS) yang berlangganan topik Amazon Simple Notification Service (Amazon SNS).

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Identitas Amazon Cognito
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Buat aplikasi yang menganalisis umpan balik pelanggan dan mensintesis audio

SDK untuk JavaScript (v3)

Aplikasi contoh ini menganalisis dan menyimpan kartu umpan balik pelanggan. Secara khusus, ini memenuhi kebutuhan hotel fiktif di New York City. Hotel menerima umpan balik dari para tamu

dalam berbagai bahasa dalam bentuk kartu komentar fisik. Umpan balik itu diunggah ke aplikasi melalui klien web. Setelah gambar kartu komentar diunggah, langkah-langkah berikut terjadi:

- Teks diekstraksi dari gambar menggunakan Amazon Textract.
- Amazon Comprehend menentukan sentimen teks yang diekstraksi dan bahasanya.
- Teks yang diekstraksi diterjemahkan ke bahasa Inggris menggunakan Amazon Translate.
- Amazon Polly mensintesis file audio dari teks yang diekstraksi.

Aplikasi lengkap dapat digunakan dengan AWS CDK Untuk kode sumber dan petunjuk penerapan, lihat proyek di [GitHub](#). Kutipan berikut menunjukkan bagaimana yang AWS SDK for JavaScript digunakan di dalam fungsi Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string }} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });
```

```
const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

return {
  sentiment: Sentiment,
  language_code: languageCode,
};
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
  eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/textract/latest/dg/API\_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );
};
```

```
    return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 * sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
    const pollyClient = new PollyClient({});

    const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
        Engine: "neural",
        Text: sourceDestinationConfig.translated_text,
        VoiceId: "Ruth",
        OutputFormat: "mp3",
    });

    const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

    const audioKey = `${sourceDestinationConfig.object}.mp3`;

    // Store the audio file in S3.
    const s3Client = new S3Client();
    const upload = new Upload({
        client: s3Client,
        params: {
            Bucket: sourceDestinationConfig.bucket,
            Key: audioKey,
            Body: AudioStream,
            ContentType: "audio/mp3",
        },
    });

    await upload.done();
    return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string }}
  textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Mendeteksi APD dalam gambar dengan Amazon AWS Rekognition menggunakan SDK

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan Amazon Rekognition dengan AWS SDK for JavaScript membuat aplikasi untuk mendeteksi alat pelindung diri (APD) pada gambar yang terletak di

bucket Amazon Simple Storage Service (Amazon S3). Aplikasi tersebut menyimpan hasilnya ke tabel Amazon DynamoDB, dan mengirimkan notifikasi email kepada admin beserta hasilnya menggunakan Amazon Simple Email Service (Amazon SES).

Pelajari cara:

- Membuat pengguna yang tidak diautentikasi menggunakan Amazon Cognito.
- Menganalisis gambar untuk APD menggunakan Amazon Rekognition.
- Memverifikasi alamat email untuk Amazon SES.
- Memperbarui tabel DynamoDB dengan hasil.
- Mengirim notifikasi email menggunakan Amazon SES.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Mendeteksi objek dalam gambar dengan Amazon Rekognition menggunakan SDK AWS

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan Amazon Rekognition dengan membuat aplikasi AWS SDK for JavaScript yang menggunakan Amazon Rekognition untuk mengidentifikasi objek berdasarkan kategori dalam gambar yang terletak di bucket Amazon Simple Storage Service (Amazon S3). Aplikasi ini mengirimkan notifikasi email kepada admin beserta hasilnya menggunakan Amazon Simple Email Service (Amazon SES).

Pelajari cara:

- Membuat pengguna yang tidak diautentikasi menggunakan Amazon Cognito.
- Menganalisis gambar untuk objek menggunakan Amazon Rekognition.

- Memverifikasi alamat email untuk Amazon SES.
- Mengirim notifikasi email menggunakan Amazon SES.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Amazon Rekognition
- Amazon S3
- Amazon SES

Mendeteksi orang dan objek dalam video dengan Amazon Rekognition menggunakan SDK AWS

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan Amazon Rekognition dengan AWS SDK for JavaScript membuat aplikasi untuk mendeteksi wajah dan objek dalam video yang terletak di bucket Amazon Simple Storage Service (Amazon S3). Aplikasi ini mengirimkan notifikasi email kepada admin beserta hasilnya menggunakan Amazon Simple Email Service (Amazon SES).

Pelajari cara:

- Membuat pengguna yang tidak diautentikasi menggunakan Amazon Cognito.
- Menganalisis gambar untuk APD menggunakan Amazon Rekognition.
- Memverifikasi alamat email untuk Amazon SES.
- Mengirim notifikasi email menggunakan Amazon SES.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Amazon Rekognition
- Amazon S3
- Amazon SES

Menginvokasi fungsi Lambda dari browser

SDK untuk JavaScript (v2)

Anda dapat membuat aplikasi berbasis browser yang menggunakan AWS Lambda fungsi untuk memperbarui tabel Amazon DynamoDB dengan pilihan pengguna.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda

SDK untuk JavaScript (v3)

Anda dapat membuat aplikasi berbasis browser yang menggunakan AWS Lambda fungsi untuk memperbarui tabel Amazon DynamoDB dengan pilihan pengguna. Aplikasi ini menggunakan AWS SDK for JavaScript v3.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda

Menggunakan API Gateway untuk menginvokasi fungsi Lambda

SDK untuk JavaScript (v3)

Menunjukkan cara membuat AWS Lambda fungsi dengan menggunakan API JavaScript runtime Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat fungsi Lambda yang diinvokasi oleh Amazon API Gateway yang memindai peringatan hari jadi kerja di tabel Amazon DynamoDB dan menggunakan Amazon Simple Notification Service (Amazon SNS) untuk mengirim pesan teks berisi ucapan selamat kepada karyawan Anda pada tanggal hari jadi kerja satu tahun mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Menggunakan Step Functions untuk menginvokasi fungsi Lambda

SDK untuk JavaScript (v3)

Menunjukkan cara membuat alur kerja AWS tanpa server dengan menggunakan AWS Step Functions dan. AWS SDK for JavaScript Setiap langkah alur kerja diimplementasikan menggunakan AWS Lambda fungsi.

Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa perlu menyediakan atau mengelola server. Step Functions adalah layanan orkestrasi nirserver yang memungkinkan Anda menggabungkan fungsi Lambda dan layanan AWS lainnya untuk membangun aplikasi bisnis penting.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda

SDK untuk JavaScript (v3)

Menunjukkan cara membuat acara EventBridge terjadwal Amazon yang memanggil AWS Lambda fungsi. Konfigurasi EventBridge untuk menggunakan ekspresi cron untuk menjadwalkan saat fungsi Lambda dipanggil. Dalam contoh ini, Anda membuat fungsi Lambda menggunakan API runtime JavaScript Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat aplikasi yang mengirimkan pesan teks seluler kepada karyawan Anda berisi ucapan selamat pada hari jadi setahun kerja mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Keamanan untuk AWS Produk atau Layanan ini

Keamanan cloud di Amazon Web Services (AWS) merupakan prioritas tertinggi. Sebagai pelanggan AWS, Anda mendapatkan manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan 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](#)
- [Manajemen Identitas dan Akses](#)
- [Validasi Kepatuhan untuk AWS Produk atau Layanan ini](#)
- [Ketahanan untuk AWS Produk atau Layanan ini](#)
- [Keamanan Infrastruktur untuk AWS Produk atau Layanan ini](#)
- [Menerapkan versi TLS minimum](#)

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 memelihara kendali atas isi yang dihost 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 kredensial 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.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-2 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi yang lebih lengkap tentang titik akhir FIPS yang tersedia di [Standar Pemrosesan Informasi Federal \(FIPS\) 140-2](#).

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 saat Anda bekerja dengan AWS produk atau layanan ini atau lainnya Layanan AWS menggunakan konsol, API AWS CLI, atau AWS SDK. 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.

Manajemen Identitas dan Akses

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 kredensial 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 Anda untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep Basic 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 kredensial 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 kredensial yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (IAM Identity Center), autentikasi masuk tunggal perusahaan Anda, dan kredensial Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas gabungan, 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 kredensial Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Untuk informasi selengkapnya tentang penggunaan metode yang disarankan untuk menandatangani permintaan sendiri, lihat [Menandatangani permintaan AWS API](#) di 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 lebih lanjut, lihat [Autentikasi multi-faktor](#) dalam Panduan Pengguna AWS IAM Identity Center dan [Menggunakan autentikasi multi-faktor \(MFA\) di AWS](#) 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 Anda. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut

untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar tugas lengkap yang mengharuskan Anda masuk sebagai pengguna root, lihat [Tugas yang memerlukan kredensial pengguna root](#) dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensial 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 kredensial yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensial sementara.

Untuk pengelolaan akses terpusat, sebaiknya 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 [Apa yang dimaksud 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, sebaiknya andalkan kredensial temporer, dan bukan membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan khusus yang memerlukan kredensial jangka panjang dengan pengguna IAM, sebaiknya rotasikan 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 kumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin untuk beberapa pengguna sekaligus. Grup membuat izin lebih mudah dikelola untuk sekelompok besar pengguna. Misalnya, Anda dapat memiliki grup yang bernama 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 tersebut dimaksudkan untuk dapat diambil oleh siapa pun yang membutuhkannya. Pengguna

memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari selengkapnya, silakan lihat [Kapan harus membuat pengguna IAM \(bukan peran\)](#) 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. Anda dapat mengambil peran IAM untuk sementara AWS Management Console dengan [beralih peran](#). Anda dapat mengambil peran dengan memanggil operasi AWS CLI atau AWS API atau dengan menggunakan URL kustom. Untuk informasi selengkapnya tentang metode untuk menggunakan peran, lihat [Menggunakan peran IAM](#) dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna gabungan – Untuk menetapkan izin ke sebuah identitas gabungan, Anda dapat membuat peran dan menentukan izin untuk peran tersebut. Saat identitas terfederasi diautentikasi, identitas tersebut dikaitkan dengan peran dan diberikan izin yang ditentukan oleh peran. Untuk informasi tentang peran untuk federasi, lihat [Membuat peran untuk Penyedia Identitas pihak ketiga](#) dalam Panduan Pengguna IAM. Jika Anda menggunakan Pusat Identitas IAM, Anda mengonfigurasi sekumpulan izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM mengaitkan izin yang ditetapkan ke peran dalam IAM. Untuk informasi tentang rangkaian izin, lihat [Rangkaian 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 (pengguna utama tepercaya) dengan akun berbeda untuk mengakses sumber daya yang ada 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 kebijakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat [Bagaimana peran IAM berbeda dari kebijakan berbasis sumber daya](#) dalam Panduan Pengguna IAM.
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Contoh, ketika Anda melakukan panggilan dalam layanan, umumnya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Suatu layanan mungkin melakukan hal

tersebut menggunakan izin pengguna utama panggilan, 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 tindakan yang kemudian memulai 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 [Meneruskan sesi akses](#).
- Peran IAM – Peran layanan adalah [peran IAM](#) yang diambil layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, memodifikasi, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat [Membuat peran untuk mendelegasikan izin ke Layanan AWS](#) dalam Panduan pengguna IAM.
- Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke peran layanan. Layanan AWS Layanan tersebut dapat mengambil peran untuk melakukan sebuah 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 kredensial sementara untuk aplikasi yang berjalan pada instans EC2 dan membuat atau permintaan API. AWS CLI AWS Cara ini lebih dianjurkan daripada menyimpan kunci akses dalam instans EC2. Untuk menetapkan AWS peran ke instans EC2 dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instans berisi peran dan memungkinkan program yang berjalan di instans EC2 mendapatkan kredensial sementara. Untuk informasi selengkapnya, lihat [Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di instans Amazon EC2](#) dalam Panduan Pengguna IAM.

Untuk mempelajari apakah kita harus menggunakan peran IAM atau pengguna IAM, lihat [Kapan harus membuat peran IAM \(bukan pengguna\)](#) dalam 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 [Ikhtisar kebijakan JSON](#) dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, pengguna utama manakah yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat menjalankan peran.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk operasi. Sebagai contoh, anggap saja 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 pengguna dan peran, di sumber daya mana, dan dengan ketentuan apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Membuat kebijakan IAM](#) dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan terkelola. 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 terkelola atau kebijakan inline, lihat [Memilih antara kebijakan terkelola 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 yang dilampiri kebijakan tersebut, kebijakan ini menentukan jenis tindakan yang dapat dilakukan oleh pengguna utama tertentu di sumber daya tersebut dan apa ketentuannya. Anda harus [menentukan pengguna utama](#) 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 (ACL)

Daftar kontrol akses (ACL) mengendalikan pengguna utama mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL sama dengan kebijakan berbasis sumber daya, meskipun tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung ACL. Untuk mempelajari ACL selengkapnya, silakan lihat [Gambaran umum daftar kontrol akses \(ACL\)](#) di Panduan Developer Layanan Penyimpanan Ringkas Amazon.

Tipe kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Tipe-tipe kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda berdasarkan tipe kebijakan yang lebih umum.

- **Batasan izin** – Batasan izin adalah fitur lanjutan di mana Anda menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM (pengguna 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 secara eksplisit terhadap salah satu kebijakan ini akan mengesampingkan izin tersebut. Untuk informasi selengkapnya tentang batasan izin, lihat [Batasan izin untuk entitas IAM](#) dalam Panduan Pengguna IAM.
- **Kebijakan kontrol layanan (SCP)** — SCP adalah kebijakan JSON yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di. AWS Organizations adalah layanan untuk mengelompokkan dan mengelola secara terpusat beberapa Akun AWS yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur dalam organisasi, Anda dapat menerapkan kebijakan kontrol layanan (SCP) ke sebagian atau semua akun Anda. SCP membatasi izin untuk entitas di akun anggota, termasuk masing-masing. Pengguna root akun AWS Untuk informasi

selengkapnya tentang Organisasi dan SCP, lihat [Cara kerja SCP](#) dalam Panduan Pengguna AWS Organizations .

- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda teruskan sebagai parameter saat Anda membuat sesi sementara secara terprogram untuk peran atau pengguna gabungan. 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 di salah satu kebijakan ini akan membatalkan izin tersebut. Untuk informasi selengkapnya, lihat [Kebijakan sesi](#) dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Jika beberapa jenis kebijakan diberlakukan untuk satu 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 `aws:GetWidget` rekaan.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
aws: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 `aws: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 pengguna 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 (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi pengguna akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa hal berikut:

- Untuk mempelajari apakah AWS mendukung fitur-fitur ini, lihat [Bagaimana 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 [Memberikan akses kepada pengguna eksternal yang sah \(federasi identitas\)](#) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara penggunaan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat [Perbedaan antara peran IAM dan kebijakan berbasis sumber daya](#) 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:

- [Panduan Memulai Cepat Keamanan dan Kepatuhan — Panduan](#) penerapan ini membahas pertimbangan arsitektur dan memberikan langkah-langkah untuk menerapkan lingkungan dasar AWS yang berfokus pada keamanan dan kepatuhan.

- [Arsitektur untuk Keamanan dan Kepatuhan HIPAA di Amazon Web Services](#) — Whitepaper ini menjelaskan bagaimana perusahaan dapat menggunakan AWS untuk membuat aplikasi yang memenuhi syarat HIPAA.

 Note

Tidak semua memenuhi Layanan AWS syarat HIPAA. Untuk informasi selengkapnya, lihat [Referensi Layanan yang Memenuhi 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 mengevaluasi sumber daya AWS Anda dan memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat [Referensi kontrol Security Hub](#).
- [AWS Audit Manager](#)Ini 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 Wilayah AWS dan Availability Zones.

Wilayah AWS 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 dengan menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan pengguna utama 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](#).

Menerapkan versi TLS minimum

Untuk menambahkan peningkatan keamanan saat berkomunikasi dengan AWS layanan, konfigurasi AWS SDK for JavaScript untuk menggunakan TLS 1.2 atau yang lebih baru.

Important

AWS SDK for JavaScript V3 secara otomatis menegosiasikan versi TLS tingkat tertinggi yang didukung oleh titik akhir Layanan tertentu AWS . Anda dapat secara opsional menerapkan versi TLS minimum yang diperlukan oleh aplikasi Anda, seperti TLS 1.2 atau 1.3, tetapi harap dicatat bahwa TLS 1.3 tidak didukung oleh beberapa titik akhir AWS Layanan, sehingga beberapa panggilan mungkin gagal jika Anda menerapkan TLS 1.3.

Transport Layer Security (TLS) adalah protokol yang digunakan oleh browser web dan aplikasi lain untuk memastikan privasi dan integritas data yang dipertukarkan melalui jaringan.

Verifikasi dan terapkan TLS di Node.js

Saat Anda menggunakan AWS SDK for 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 for JavaScript v3 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.

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 11.0.0 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 menginstal dan memperbarui](#). Kemudian jalankan perintah berikut untuk menginstal dan menggunakan versi tertentu dari Node.js.

```
nvm install 11  
nvm use 11
```

Enforce 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.

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
  region: "us-west-2",
  requestHandler: new NodeHttpHandler({
    httpsAgent: new https.Agent({
      {
        secureProtocol: 'TLSv1_2_method'
      }
    })
  })
});
```

Enforce TLS 1.3

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.

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
  region: "us-west-2",
  requestHandler: new NodeHttpHandler({
    httpsAgent: new https.Agent({
      {
        secureProtocol: 'TLSv1_3_method'
      }
    })
  })
});
```

```
    })  
  });
```

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 *opsi Internet*.
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.

Migrasi ke versi 3

Bagian ini menjelaskan cara bermigrasi dari versi 2 ke versi 3. AWS SDK for JavaScript

Migrasikan kode Anda ke SDK for V3 JavaScript

AWS SDK for JavaScript versi 3 (v3) dilengkapi dengan antarmuka modern untuk konfigurasi dan utilitas klien, yang mencakup kredensial, unggahan multipart Amazon S3, klien dokumen DynamoDB, pelayan, dan banyak lagi. Anda dapat menemukan apa yang berubah di v2 dan setara v3 untuk setiap perubahan dalam [panduan migrasi di repo](#). AWS SDK for JavaScript GitHub

Untuk memanfaatkan sepenuhnya AWS SDK for JavaScript v3, kami sarankan menggunakan skrip codemod yang dijelaskan di bawah ini.

Gunakan codemod untuk memigrasikan kode v2 yang ada

Kumpulan skrip codemod di [aws-sdk-js-codemod](#) membantu memigrasikan aplikasi AWS SDK for JavaScript (v2) Anda yang ada untuk menggunakan API v3. Anda dapat menjalankan transformasi sebagai berikut.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

Misalnya, pertimbangkan Anda memiliki kode berikut, yang membuat klien Amazon DynamoDB dari v2 dan operasi panggilan. `listTables`

```
// example.ts
import AWS from "aws-sdk";

const region = "us-west-2";
const client = new AWS.DynamoDB({ region });
client.listTables({}, (err, data) => {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

Anda dapat menjalankan `v2-to-v3` transformasi kami `example.ts` sebagai berikut.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

Transformasi akan mengonversi impor DynamoDB ke v3, membuat klien v3 dan memanggil operasi sebagai berikut. `listTables`

```
// example.ts
import { DynamoDB } from "@aws-sdk/client-dynamodb";

const region = "us-west-2";
const client = new DynamoDB({ region });
client.listTables({}, (err, data) => {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

Kami telah menerapkan transformasi untuk kasus penggunaan umum. Jika kode Anda tidak berubah dengan benar, buat [laporan bug](#) atau [permintaan fitur](#) dengan contoh kode input dan kode keluaran yang diamati/diharapkan. Jika kasus penggunaan spesifik Anda sudah dilaporkan dalam [masalah yang ada](#), tunjukkan dukungan Anda melalui upvote.

Riwayat dokumen untuk AWS SDK for JavaScript versi 3

Riwayat Dokumen

Tabel berikut menjelaskan perubahan penting dalam rilis V3 AWS SDK for JavaScript dari 20 Oktober 2020, dan seterusnya. Untuk notifikasi tentang pembaruan dokumentasi ini, Anda dapat berlangganan ke [umpan RSS](#).

Perubahan	Deskripsi	Tanggal
Pengumuman	Spanduk teratas yang diperbarui dengan end-of-support pengingat untuk Internet Explorer 11.	September 23, 2022
Pembaruan kecil	Pembaruan kecil untuk kejelasan dan penyelesaian tautan yang rusak. Menambahkan tautan kesadaran ke AWS SDK dan Panduan Referensi Alat.	22 Agustus 2022
Menerapkan versi TLS minimum	Menambahkan informasi tentang TLS 1.3.	31 Maret 2022
AWS Lambda Tutorial yang diperbarui	Ditambahkan tutorial yang menunjukkan cara membangun aplikasi berbasis browser untuk mengirimkan data ke tabel Amazon DynamoDB.	20 Oktober 2020
Mengatur kredensial dalam topik Node.js diperbarui	Perbarui topik tentang pengaturan kredensial di Node.js untuk AWS SDK for JavaScript V3.	20 Oktober 2020

Migrasi ke V3	Menambahkan topik untuk menjelaskan cara bermigrasi ke AWS SDK for JavaScript V3.	20 Oktober 2020
Memulai	Topik yang diperbarui untuk memulai di browser dan memulai dengan Node.js untuk AWS SDK for JavaScript V3.	20 Oktober 2020
Pembuat peramban	Informasi tentang AWS Browser Builder telah dihapus karena tidak diperlukan untuk AWS SDK for JavaScript V3.	20 Oktober 2020
Contoh layanan Amazon Transcribe diperbarui	Contoh layanan Amazon Transcribe yang diperbarui untuk AWS SDK for JavaScript V3.	20 Oktober 2020
Contoh layanan Layanan Pemberitahuan Sederhana Amazon diperbarui	Diperbarui Amazon Simple Notification Service contoh layanan untuk AWS SDK for JavaScript V3.	20 Oktober 2020
Contoh layanan Layanan Email Amazon Simple diperbarui	Diperbarui Amazon Simple Email Service contoh layanan untuk AWS SDK for JavaScript V3.	20 Oktober 2020
Contoh layanan Amazon Redshift diperbarui	Contoh layanan Amazon Redshift yang diperbarui untuk AWS SDK for JavaScript V3.	20 Oktober 2020
Contoh layanan Amazon Lex diperbarui	Contoh layanan Amazon Lex yang diperbarui untuk AWS SDK for JavaScript V3.	20 Oktober 2020

Contoh layanan Amazon DynamoDB diperbarui	Diperbarui Amazon DynamoDB contoh layanan untuk V3. AWS SDK for JavaScript	20 Oktober 2020
AWS Elemental MediaConvert contoh layanan diperbarui	Contoh AWS Elemental MediaConvert layanan yang diperbarui untuk AWS SDK for JavaScript V3.	20 Oktober 2020
AWS Lambda contoh layanan diperbarui	Contoh AWS Lambda layanan yang diperbarui untuk AWS SDK for JavaScript V3.	20 Oktober 2020
AWS SDK for JavaScript Pratinjau Panduan Pengembang V3	Versi pra-rilis yang dirilis dari Panduan Pengembang AWS SDK for JavaScript V3.	19 Oktober 2023