SDK 버전 3용 개발자 안내서

AWS SDK for JavaScript

AWS SDK for JavaScript: SDK 버전 3용 개발자 안내서

Table of Contents

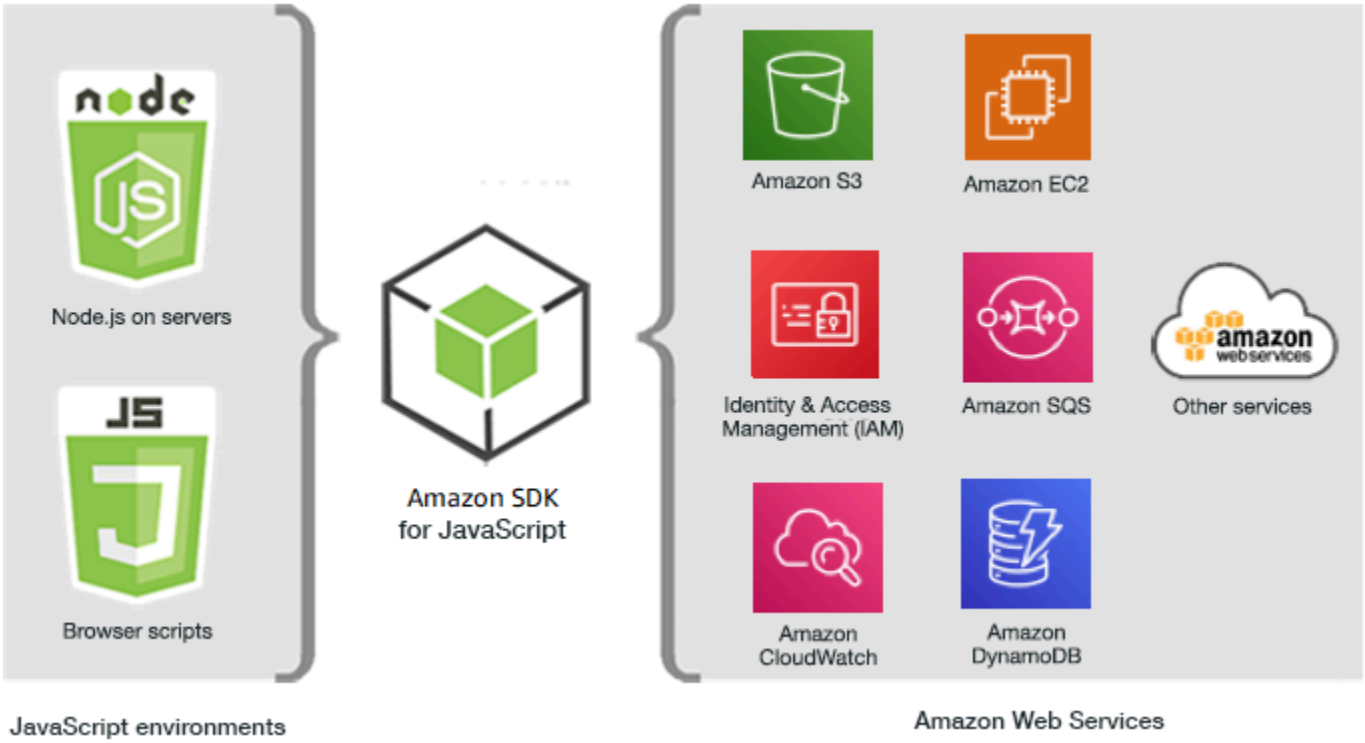
AWS SDK for JavaScript V3 API 참조 안내서는 AWS SDK for JavaScript 버전 3(V3)의 모든 API 작업을 자세히 설명합니다.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

AWS SDK for JavaScript란 무엇인가요?

AWS SDK for JavaScript 개발자 안내서에 오신 것을 환영합니다. 이 안내서는 AWS SDK for JavaScript 설정 및 구성에 관한 일반적인 정보를 제공합니다. 또한 AWS SDK for JavaScript를 사용하여 다양한 AWS 서비스를 실행하는 예와 자습서를 안내합니다.

AWS SDK for JavaScript v3 API Reference Guide에서는 AWS 서비스용 JavaScript API를 제공합니다. JavaScript API를 사용하여 Node.js용 또는 브라우저용 라이브러리 또는 애플리케이션을 빌드할 수 있습니다.



SDK 메이저 버전에 대한 유지 관리 및 지원

SDK 메이저 버전 및 기본 종속성의 유지 관리 및 지원에 대한 자세한 내용은 AWS SDK 및 도구 참조 안내서에서 다음 내용을 참조하세요.

- AWS SDK 및 도구 유지 관리 정책
- AWS SDK 및 도구 버전 지원 매트릭스

버전 3의 새 기능

SDK for JavaScript 버전 3(V3)에는 다음과 같은 새로운 기능이 포함되어 있습니다.

모듈화된 패키지

이제 사용자는 각 서비스에 대해 별도의 패키지를 사용할 수 있습니다.

새 미들웨어 스택

이제 사용자는 미들웨어 스택을 사용하여 작업 호출의 수명 주기를 제어할 수 있습니다.

또한 SDK는 TypeScript로 작성되어 정적 형식 지정 등 많은 장점이 있습니다.

> **Important**
>
> 이 안내서의 V3에 대한 코드 예는 ECMAScript 6(ES6)로 작성되었습니다. ES6는 코드를 더 현대적이고 읽기 쉽게 만들고 더 많은 작업을 수행할 수 있도록 새로운 구문과 새로운 기능을 제공합니다. ES6에서는 Node.js 버전 13.x 이상을 사용해야 합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 자세한 내용은 JavaScript ES6/CommonJS 구문 섹션을 참조하세요.

모듈화된 패키지

SDK for JavaScript 버전 2(V2)에서는 다음과 같이 전체 AWS SDK를 사용해야 했습니다.

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

애플리케이션이 많은 AWS 서비스를 사용하는 경우 전체 SDK를 로드하는 것은 문제가 되지 않습니다. 그러나 AWS 서비스를 몇 가지만 사용해야 하는 경우 전체 SDK를 로드하는 것은 필요하지 않거나 사용하지 않는 코드로 애플리케이션의 크기를 늘리는 것을 의미합니다.

V3에서는 필요한 개별 AWS 서비스만 로드하여 사용할 수 있습니다. 이는 다음 예에 나와 있는데, 이렇게 하면 Amazon DynamoDB(DynamoDB)에 액세스할 수 있습니다.

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

개별 AWS 서비스를 로드하여 사용할 수 있을 뿐만 아니라 필요한 서비스 명령만 로드하여 사용할 수도 있습니다. 이는 DynamoDB 클라이언트 및 `ListTablesCommand` 명령에 액세스할 수 있는 다음 예에 나와 있습니다.

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

Important

하위 모듈을 모듈로 가져오면 안 됩니다. 예를 들어 다음 코드에서는 오류가 발생할 수 있습니다.

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

다음은 올바른 코드입니다.

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

코드 크기 비교

버전 2(V2)에서 `us-west-2` 리전의 모든 Amazon DynamoDB 테이블을 나열하는 간단한 코드 예는 다음과 같을 수 있습니다.

```
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);
  }
```

```
});
```

V3에서는 코드가 다음과 같습니다.

```
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-sdk 패키지는 애플리케이션에 약 40MB를 추가합니다. `var AWS = require("aws-sdk")`를 `import {DynamoDB} from "@aws-sdk/client-dynamodb"`로 바꾸면 오버헤드가 약 3MB로 줄어듭니다. 가져오기를 DynamoDB 클라이언트 및 `ListTablesCommand` 명령으로만 제한하면 오버헤드가 100KB 미만으로 줄어듭니다.

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

V3에서 명령 호출

V3에서 V2 또는 V3 명령을 사용하여 작업을 수행할 수 있습니다. V3 명령을 사용하려면 명령과 필수 AWS 서비스 패키지 클라이언트를 가져오고 async/await 패턴을 사용하는 `.send` 메서드를 사용하여 명령을 실행합니다.

V2 명령을 사용하려면 필수 AWS 서비스 패키지를 가져오고 콜백 또는 async/await 패턴을 사용하여 패키지에서 직접 V2 명령을 실행합니다.

V3 명령 사용

V3에서는 각 AWS 서비스 패키지의 명령 집합을 제공하여 해당 AWS 서비스에 대한 작업을 수행할 수 있도록 지원합니다. AWS 서비스를 설치한 후 프로젝트의 `node-modules/@aws-sdk/client-`*`PACKAGE_NAME`*`/commands folder.`에서 사용 가능한 명령을 찾아볼 수 있습니다.

사용하려는 명령을 가져와야 합니다. 예를 들어 다음 코드는 DynamoDB 서비스와 `CreateTableCommand` 명령을 로드합니다.

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

권장되는 async/await 패턴으로 이러한 명령을 직접적으로 호출하려면 다음 구문을 사용합니다.

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

예를 들어 다음 예에서는 권장되는 async/await 패턴을 사용하여 DynamoDB 테이블을 생성합니다.

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

V2 명령 사용

SDK for JavaScript에서 V2 명령을 사용하려면 다음 코드에 나와 있는 대로 전체 AWS 서비스 패키지를 가져옵니다.

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

권장되는 async/await 패턴으로 V2 명령을 직접적으로 호출하려면 다음 구문을 사용합니다.

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

다음 예에서는 V2 `createTable` 명령을 사용하여 권장되는 async/await 패턴으로 DynamoDB 테이블을 생성합니다.

```
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();
```

다음 예에서는 V2 `createBucket` 명령을 사용하여 콜백 패턴으로 Amazon S3 버킷을 생성합니다.

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

새 미들웨어 스택

SDK V2를 사용하면 요청에 이벤트 리스너를 연결하여 수명 주기의 여러 단계에 걸쳐서 요청을 수정할 수 있었습니다. 이 접근 방식을 사용하면 요청의 수명 주기 동안 무엇이 잘못되었는지 디버깅하기가 어려울 수 있습니다.

V3에서는 새 미들웨어 스택을 사용하여 작업 호출의 수명 주기를 제어할 수 있습니다. 이 접근 방식은 몇 가지 이점을 제공합니다. 스택의 각 미들웨어 단계는 요청 객체를 변경한 후 다음 미들웨어 단계를 직접적으로 호출합니다. 이렇게 하면 어떤 미들웨어 단계가 호출되어 오류가 발생했는지 정확하게 확인할 수 있으므로 스택의 문제를 디버깅하는 것도 훨씬 쉬워집니다.

다음 예에서는 미들웨어를 사용하여 Amazon DynamoDB 클라이언트(앞서 생성하고 보여준)에 사용자 지정 헤더를 추가합니다. 첫 번째 인수는 직접적으로 호출할 스택의 다음 미들웨어 단계인 `next`와 직접적으로 호출되는 작업에 관한 일부 정보가 포함된 객체인 `context`를 받는 함수입니다. 이 함수는 작업 및 요청에 전달되는 파라미터가 포함된 객체인 `args`를 받는 함수를 반환합니다. `args`를 사용하여 다음 미들웨어를 호출한 결과를 반환합니다.

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

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

Node.js에서 SDK 사용

Node.js는 서버 측 JavaScript 애플리케이션을 실행하기 위한 교차 플랫폼 런타임입니다. Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스에 Node.js를 설정하여 서버에서 실행할 수 있습니다. 또한 Node.js를 사용하여 온디맨드 AWS Lambda 함수를 작성할 수도 있습니다.

Node.js에 SDK를 사용하는 것은 웹 브라우저에서 JavaScript에 SDK를 사용하는 방법과 다릅니다. SDK를 로드하고 특정 웹 서비스에 액세스하는 데 필요한 자격 증명을 얻는 방법에서 차이가 비롯됩니다. 특정 API 사용이 Node.js와 브라우저 간에 다른 경우 해당 차이점을 표시합니다.

AWS Cloud9에서 SDK 사용

AWS Cloud9 IDE에서 SDK for JavaScript를 사용하여 Node.js 애플리케이션을 개발할 수도 있습니다. SDK for JavaScript와 함께 AWS Cloud9을 사용하는 방법에 관한 자세한 내용은 AWS Cloud9 와 함께 사용 AWS SDK for JavaScript 단원을 참조하세요.

AWS Amplify에서 SDK 사용

브라우저 기반 웹, 모바일 및 하이브리드 앱의 경우GitHub의 AWS Amplify 라이브러리를 사용할 수도 있습니다. Amplify는 SDK for JavaScript를 확장하여 선언적 인터페이스를 제공합니다.

> **Note**
>
> Amplify와 같은 프레임워크는 SDK for JavaScript와 동일한 브라우저 지원을 제공하지 않을 수 있습니다. 자세한 내용은 프레임워크의 설명서를 참조하세요.

웹 브라우저에서 SDK 사용

주요 웹 브라우저는 모두 JavaScript 확장을 지원합니다. 웹 브라우저에서 실행 중인 JavaScript 코드를 일반적으로 클라이언트 측 JavaScript라고 합니다.

AWS SDK for JavaScript에서 지원되는 브라우저 목록은 지원되는 웹 브라우저 섹션을 참조하십시오.

웹 브라우저에서 SDK for JavaScript를 사용하는 방법은 Node.js에 SDK를 사용하는 방법과 다릅니다. SDK를 로드하고 특정 웹 서비스에 액세스하는 데 필요한 자격 증명을 얻는 방법에서 차이가 비롯됩니다. 특정 API 사용이 Node.js와 브라우저 간에 다른 경우 해당 차이점을 표시합니다.

V3에서 브라우저 사용

V3를 사용하면 필요한 SDK for JavaScript 파일만 번들로 제공하고 브라우저에 포함할 수 있으므로 오버헤드가 줄어듭니다.

HTML 페이지에서 SDK for JavaScript V3를 사용하려면 WebPack을 사용하여 필수 클라이언트 모듈과 모든 필수 JavaScript 함수를 단일 JavaScript 파일로 번들링하고 이를 HTML 페이지의 <head>에 있는 스크립트 태그에 추가해야 합니다. 예:

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

> **Note**
>
> Webpack에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

SDK for JavaScript V2를 사용하려면 대신 최신 버전의 V2 SDK를 가리키는 스크립트 태그를 추가합니다. 자세한 내용은 AWS SDK for JavaScript v2 개발자 안내서의 sample 단원을 참조하세요.

일반 사용 사례

브라우저 스크립트에서 SDK for JavaScript를 사용하면 여러 가지 흥미로운 사용 사례를 실현할 수 있습니다. 다음은 SDK for JavaScript를 사용하여 다양한 웹 서비스에 액세스함으로써 브라우저 애플리케이션에서 빌드할 수 있는 것에 대한 몇 가지 아이디어입니다.

- 조직 또는 프로젝트 요구 사항을 최대한 충족하기 위해 리전 및 서비스 전반에 걸쳐 기능에 액세스하고 기능을 결합하는 사용자 지정 콘솔을 AWS 서비스에 빌드합니다.
- Amazon Cognito 자격 증명을 사용하여 인증된 사용자가 Facebook 등의 타사 인증 사용을 포함해 브라우저 애플리케이션 및 웹 사이트에 액세스하도록 합니다.
- Amazon Kinesis를 사용하여 클릭 스트림 또는 기타 마케팅 데이터를 실시간으로 처리합니다.
- 웹 사이트 방문자 또는 애플리케이션 사용자에 대한 개별 사용자 기본 설정과 같은 서버리스 데이터 지속성에 Amazon DynamoDB를 사용합니다.
- AWS Lambda를 사용하여 지적 재산을 다운로드해 사용자에게 노출하는 일 없이 브라우저 스크립트에서 호출할 수 있는 독점 로직을 캡슐화합니다.

예시 관련 정보

SDK for JavaScript 예는 AWS Code Example Repository에서 찾아볼 수 있습니다.

리소스

SDK for JavaScript 개발자는 이 안내서 외에도 다음과 같은 온라인 리소스를 사용할 수 있습니다.

- AWS SDK for JavaScript V3 API 참조 가이드
- AWS SDK 및 도구 참조 가이드: AWS SDK에서 흔히 사용되는 설정, 기능 및 기타 기본 개념이 포함되어 있습니다.
- JavaScript 개발자 블로그

- AWS JavaScript 포럼
- AWS코드 카탈로그의 JavaScript 예
- AWS Code Example Repository
- Gitter 채널
- Stack Overflow
- AWS-sdk-js로 태그가 지정된 Stack Overflow 질문
- GitHub
  - SDK Source
  - Documentation Source

AWS SDK for JavaScript 시작

AWS SDK for JavaScript는 브라우저 또는 Node.js 환경에서 웹 서비스에 대한 액세스를 제공합니다. 이 단원에는 이러한 각 JavaScript 환경에서 SDK for JavaScript를 사용하여 작업하는 방법을 보여주는 시작하기 연습이 있습니다.

> **Note**
>
> AWS Cloud9 IDE에서 SDK for JavaScript를 사용하여 브라우저 기반 애플리케이션을 위한 JavaScript 및 Node.js 애플리케이션을 개발할 수 있습니다. Node.js 개발에 AWS Cloud9을 사용하는 방법의 예는 AWS Cloud9 와 함께 사용 AWS SDK for JavaScript 단원을 참조하세요.

주제

- AWS를 사용한 SDK 인증
- Node.js 시작하기
- 브라우저에서 시작하기

AWS를 사용한 SDK 인증

AWS 서비스로 개발할 때는 코드가 AWS에서 인증되는 방법을 설정해야 합니다. 환경 및 사용 가능한 AWS 액세스 권한에 따라 다양한 방식으로 AWS 리소스에 대한 프로그래밍 방식의 액세스를 구성할 수 있습니다.

인증 방법을 선택하고 SDK에 맞게 구성하려면 AWS SDK 및 도구 참조 가이드의 Authentication and access 단원을 참조하세요.

로컬에서 개발 중이고 고용주로부터 인증 방법을 제공받지 못한 신규 사용자는 AWS IAM Identity Center을 설정하는 것이 좋습니다. 이 방법에는 구성을 쉽게 하고 AWS 액세스 포털에 정기적으로 로그인하기 위한 AWS CLI 설치가 포함됩니다. 이 방법을 선택하는 경우 AWSSDK 및 도구 참조 가이드의 IAM Identity Center 인증 절차를 완료한 후 환경에 다음 요소를 포함해야 합니다.

- AWS CLI는 애플리케이션을 실행하기 전에 AWS 액세스 포털 세션을 시작하는 데 사용합니다.
- SDK에서 참조할 수 있는 구성 값 세트가 포함된 `[default]` 프로필이 있는 공유 AWSconfig 파일. 이 파일의 위치를 찾으려면 AWS SDK 및 도구 참조 가이드에서 공유 파일의 위치를 참조하세요.

- 공유 `config` 파일은 `region` 설정을 지정합니다. 이는 SDK가 AWS 요청에 사용하는 기본 AWS 리전을 설정합니다. 이 리전은 사용할 리전이 지정되지 않은 SDK 서비스 요청에 사용됩니다.
- SDK는 AWS에 요청을 보내기 전에 프로필의 SSO 토큰 공급자 구성을 사용하여 보안 인증을 얻습니다. `sso_role_name` 값은 IAM Identity Center 권한 집합에 연결된 IAM 역할로, 애플리케이션에서 사용되는 AWS 서비스에 대한 액세스를 허용합니다.

  다음 샘플 `config` 파일은 SSO 토큰 공급자 구성으로 설정된 기본 프로필을 보여줍니다. 프로필의 `sso_session` 설정은 이름이 지정된 `sso-session` 섹션을 참조합니다. `sso-session` 섹션에는 AWS 액세스 포털 세션을 시작하기 위한 설정이 포함되어 있습니다.

```
[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 JavaScript v3에서는 IAM Identity Center 인증을 사용하기 위해 애플리케이션에 부가적인 패키지(예: `SSO` 및 `SSOOIDC`)를 추가할 필요가 없습니다.

이 보안 인증 공급자를 명시적으로 사용하는 방법에 대한 자세한 내용은 npm(Node.js 패키지 관리자) 웹 사이트에서 `fromSSO()`를 참조하세요.

AWS 액세스 포털 세션 시작

AWS 서비스에 액세스하는 애플리케이션을 실행하기 전에 SDK가 IAM Identity Center 인증을 사용하여 보안 인증을 확인하려면 활성화된 AWS 액세스 포털 세션이 있어야 합니다. 구성된 세션 길이에 따라 결국 액세스가 만료되고 SDK에 인증 오류가 발생합니다. AWS 액세스 포털에 로그인하려면 AWS CLI에서 다음 명령을 실행합니다.

```
aws sso login
```

지침에 따라 기본 프로필을 설정했다면 `--profile` 옵션으로 명령을 직접적으로 호출할 필요가 없습니다. SSO 토큰 공급자 구성에서 명명된 프로필을 사용하는 경우 `aws sso login --profile named-profile` 명령을 사용합니다.

필요할 경우 이미 활성 세션이 있는지 테스트하려면 다음 AWS CLI 명령을 실행합니다.

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

세션이 활성 상태인 경우 이 명령에 대한 응답은 공유 `config` 파일에 구성된 IAM Identity Center 계정 및 권한 집합을 보고합니다.

> **Note**
>
> 이미 활성 AWS 액세스 포털 세션이 있고 `aws sso login`을 실행하는 경우 보안 인증 정보를 제공하지 않아도 됩니다.
> 로그인 과정에서 데이터 AWS CLI 액세스를 허용하라는 메시지가 표시될 수 있습니다. AWS CLI는 SDK for Python를 기반으로 구축되므로 권한 메시지에는 `botocore` 이름의 변형이 포함될 수 있습니다.

세부 인증 정보

인간 사용자(인간 ID라고도 함)는 애플리케이션의 사용자, 관리자, 개발자, 운영자 및 소비자입니다. AWS 환경 및 애플리케이션에 액세스하려면 ID가 있어야 합니다. 조직의 구성원인 인간 사용자, 즉 개발자는 작업 인력 ID라고도 합니다.

AWS에 액세스할 때 임시 보안 인증을 사용합니다. 임시 보안 인증을 제공하는 역할을 가정하여 인간 사용자가 AWS 계정에 페더레이션 액세스를 제공하도록 ID 공급자를 사용할 수 있습니다. 중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center(IAM Identity Center)을 사용하여 계정에 대한 액세스와 해당 계정 내 권한을 관리하는 것이 좋습니다. 더 많은 대안을 보려면 다음을 참조하세요.

- 모범 사례에 대해 자세히 알아보려면 IAM 사용 설명서에서 IAM의 보안 모범 사례를 참조하세요.
- 단기 AWS 보안 인증 정보를 만들려면 IAM 사용 설명서에서 임시 보안 인증 정보를 참조하세요.
- 다른 AWS SDK for JavaScript V3 보안 인증 공급자에 관해 알아보려면 AWS SDK 및 도구 참조 가이드의 Standardized credential providers 단원을 참조하세요.

Node.js 시작하기

이 안내서에서는 NPM 패키지를 초기화하고 패키지에 서비스 클라이언트를 추가하며 JavaScript SDK를 사용하여 서비스 작업을 직접적으로 호출하는 방법을 보여줍니다.

시나리오

다음을 수행하는 하나의 기본 파일을 사용하여 새 NPM 패키지를 생성합니다.

- Amazon Simple Storage Service 버킷 생성
- Amazon S3 버킷에 객체 배치
- Amazon S3 버킷의 객체 읽기
- 사용자가 리소스 삭제를 원하는지 확인

사전 조건

예를 실행하려면 먼저 다음을 수행해야 합니다.

- SDK 인증을 구성합니다. 자세한 내용은 AWS를 사용한 SDK 인증 섹션을 참조하세요.
- Node.js를 설치합니다.

1단계: 패키지 구조 설정 및 클라이언트 패키지 설치

패키지 구조를 설정하고 클라이언트 패키지를 설치하려면 다음을 수행합니다.

1. 패키지를 포함할 새 `nodegetstarted` 폴더를 생성합니다.
2. 명령줄에서 새 폴더로 이동합니다.
3. 다음 명령을 실행하여 기본 `package.json` 파일을 생성합니다.

```
npm init -y
```

4. 다음 명령을 실행하여 Amazon S3 클라이언트 패키지를 설치합니다.

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

5. `package.json` 파일에 `"type": "module"`을 추가합니다. 이렇게 하면 Node.js가 최신 ESM 구문을 사용하게 됩니다. 최종 `package.json`은 다음과 비슷해야 합니다.

```
{
  "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"
}
```

2단계: 필요한 가져오기 및 SDK 코드 추가

nodegetstarted 폴더의 index.js라는 파일에 다음 코드를 추가합니다.

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

코드 예는 여기 GitHub에서 찾을 수 있습니다.

3단계: 예시 실행

> **Note**
>
> 로그인하는 것을 잊지 마세요! IAM Identity Center를 사용하여 인증하는 경우 AWS CLI `aws sso login` 명령을 사용하여 로그인해야 합니다.

1. `node index.js`를 실행합니다.
2. 버킷을 비우고 삭제할지 여부를 선택합니다.
3. 버킷을 삭제하지 않는 경우 나중에 수동으로 비우고 삭제해야 합니다.

브라우저에서 시작하기

이 단원에서는 브라우저에서 SDK for JavaScript 버전 3(V3)을 실행하는 방법을 보여주는 예를 살펴봅니다.

> **Note**
>
> 브라우저에서 V3를 실행하는 것은 버전 2(V2)와 약간 다릅니다. 자세한 내용은 V3에서 브라우저 사용 섹션을 참조하세요.

SDK for JavaScript(V3) 사용의 다른 예는 JavaScript (v3) 용 SDK 코드 예제 단원을 참조하세요.

이 웹 애플리케이션 예는 다음을 보여줍니다.

- 인증에 Amazon Cognito를 사용하여 AWS 서비스에 액세스하는 방법
- AWS Identity and Access Management(IAM) 역할을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷의 객체 목록을 읽는 방법

> **Note**
>
> 이 예에서는 인증에 AWS IAM Identity Center를 사용하지 않습니다.

시나리오

Amazon S3는 업계 최고의 확장성, 데이터 가용성, 보안 및 성능을 제공하는 객체 스토리지 서비스입니다. Amazon S3를 사용하여 버킷이라는 컨테이너 내에 객체로 데이터를 저장할 수 있습니다. Amazon S3에 관한 자세한 내용은 Amazon S3 사용 설명서를 참조하세요.

이 예는 Amazon S3 버킷에서 읽을 IAM 역할을 맡는 웹 앱을 설정하고 실행하는 방법을 보여줍니다. 이 예에서는 React 프런트엔드 라이브러리와 Vite 프런트엔드 도구를 사용하여 JavaScript 개발 환경을 제공합니다. 웹 앱은 Amazon Cognito 자격 증명 풀을 사용하여 AWS 서비스에 액세스하는 데 필요한 보안 인증을 제공합니다. 포함된 코드 예는 웹 앱에서 SDK for JavaScript를 로드하고 사용하는 기본 패턴을 보여줍니다.

1단계: Amazon Cognito 자격 증명 풀 및 IAM 역할 생성

이 연습에서는 Amazon Cognito 자격 증명 풀을 생성해서 사용하여 Amazon S3 서비스용 웹 앱에 대한 미인증 액세스 권한을 제공합니다. 자격 증명 풀을 생성하면 미인증 게스트 사용자를 지원하는 AWS Identity and Access Management(IAM) 역할도 생성됩니다. 이 예에서는 작업에 집중할 수 있도록 미인증 사용자 역할만 사용해 작업합니다. 자격 증명 공급자 및 인증된 사용자에 대한 지원은 나중에 통합할 수 있습니다. Amazon Cognito 자격 증명 풀 추가에 관한 자세한 내용은 Amazon Cognito 개발자 안내서의 자습서: 자격 증명 풀 생성 단원을 참조하세요.

Amazon Cognito 자격 증명 풀 및 연결된 IAM 역할 생성 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/cognito/에서 Amazon Cognito 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 자격 증명 풀을 선택합니다.
3. 자격 증명 풀 생성을 선택합니다.
4. 자격 증명 풀 신뢰 구성에서 사용자 인증을 위한 게스트 액세스를 선택합니다.
5. 권한 구성에서 새 IAM 역할 생성을 선택하고 IAM 역할 이름에 이름(예: getStartedRole)을 입력합니다.
6. 속성 구성에서 자격 증명 풀 이름에 이름(예: getStartedPool)을 입력합니다.
7. 검토 및 생성에서 새 자격 증명 풀에 대한 선택 사항을 확인합니다. 편집을 선택하여 마법사로 돌아가서 설정을 변경합니다. 완료하면 자격 증명 풀 생성을 선택합니다.
8. 새로 생성된 Amazon Cognito 자격 증명 풀의 자격 증명 풀 ID 및 리전을 기록해 둡니다. 이러한 값은 4단계: 브라우저 코드 설정에서 *IDENTITY_POOL_ID* 및 *REGION*을 바꾸는 데 필요합니다.

Amazon Cognito 자격 증명 풀을 생성하면 웹 앱에 필요한 Amazon S3에 대한 권한을 추가할 준비가 된 것입니다.

2단계: 생성된 IAM 역할에 정책 추가

웹 앱에서 Amazon S3 버킷에 대한 액세스를 활성화하려면 Amazon Cognito 자격 증명 풀(예: getStartedPool)에 대해 생성한 미인증 IAM 역할(예: getStartedRole)을 사용합니다. 이렇게 하려면 역할에 IAM 정책을 연결해야 합니다. IAM 역할 수정에 관한 자세한 내용은 IAM 사용 설명서의 역할 권한 정책 수정 단원을 참조하세요.

미인증 사용자와 연결된 IAM 역할에 Amazon S3 정책을 추가하는 방법

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할(Roles)을 선택합니다.
3. 수정하려는 역할의 이름(예: getStartedRole)을 선택한 다음, 권한 탭을 선택합니다.
4. 권한 추가를 선택한 다음, 정책 연결을 선택합니다.
5. 이 역할의 권한 추가 페이지에서 AmazonS3ReadOnlyAccess 확인란을 찾아 선택합니다.

   > **Note**
   >
   > 이 프로세스를 사용하여 AWS 서비스에 대한 액세스를 활성화할 수 있습니다.

6. 권한 추가를 선택합니다.

Amazon Cognito 자격 증명 풀을 생성하고 미인증 사용자의 IAM 역할에 Amazon S3에 대한 권한을 추가하면 Amazon S3 버킷을 추가하고 구성할 준비가 된 것입니다.

3단계: Amazon S3 버킷 및 객체 추가

이 단계에서는 예를 위해 Amazon S3 버킷 및 객체를 추가합니다. 또한 버킷에 대해 교차 오리진 리소스 공유(CORS)를 활성화합니다. Amazon S3 버킷 및 객체 생성에 관한 자세한 내용은 Amazon S3 사용 설명서의 Amazon S3 시작하기 단원을 참조하세요.

CORS를 사용하여 Amazon S3 버킷 및 객체를 추가하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 버킷을 선택한 다음, 버킷 생성을 선택합니다.
3. 버킷 이름 지정 규칙(예: getstartedbucket)을 준수하는 버킷 이름을 입력하고 버킷 생성을 선택합니다.
4. 생성한 버킷을 선택한 다음, 객체 탭을 선택합니다. 그런 다음, 업로드를 선택합니다.
5. 파일 및 폴더(Files and folders)에서 파일 추가(Add files)를 선택합니다.
6. 업로드할 파일을 선택한 후 열기를 선택합니다. 그런 다음, 업로드를 선택하여 버킷에 객체 업로드 작업을 완료합니다.

7. 다음으로, 버킷의 권한 탭을 선택한 다음, 교차 오리진 리소스 공유(CORS) 섹션에서 편집을 선택합니다. 다음 JSON을 입력합니다.

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

8. 변경 사항 저장을 선택합니다.

Amazon S3 버킷과 객체를 추가했다면 브라우저 코드를 설정할 준비가 된 것입니다.

4단계: 브라우저 코드 설정

애플리케이션 예는 단일 페이지의 React 애플리케이션으로 구성되어 있습니다. 이 예를 위한 파일은 여기 GitHub에서 찾을 수 있습니다.

애플리케이션 예를 설정하는 방법

1. Node.js를 설치합니다.
2. 명령줄에서 AWS 코드 예 리포지토리를 복제합니다.

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

3. 다음과 같이 애플리케이션 예로 이동합니다.

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

4. 다음 명령을 실행하여 필수 패키지를 설치합니다.

```
npm install
```

5. 다음으로, 텍스트 편집기에서 `src/App.tsx`를 열고 다음을 완료합니다.

   - *YOUR_IDENTITY_POOL_ID*를 1단계: Amazon Cognito 자격 증명 풀 및 IAM 역할 생성에서 기록해 둔 Amazon Cognito 자격 증명 풀 ID로 바꿉니다.
   - 리전 값을 Amazon S3 버킷 및 Amazon Cognito 자격 증명 풀에 할당된 리전으로 바꿉니다. 두 서비스의 리전은 모두 동일해야 합니다(예: us-east-2).
   - *bucket-name*을 3단계: Amazon S3 버킷 및 객체 추가에서 생성한 버킷 이름으로 바꿉니다.

텍스트를 바꿨다면 `App.tsx` 파일을 저장합니다. 이제 웹 앱을 실행할 준비가 되었습니다.

5단계: 예 실행

애플리케이션 예를 실행하는 방법

1. 명령줄에서 다음과 같이 애플리케이션 예로 이동합니다.

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

2. 명령줄에서 다음 명령을 실행합니다.

```
npm run dev
```

   Vite 개발 환경은 다음 메시지와 함께 실행됩니다.

```
VITE v4.3.9  ready in 280 ms

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

3. 웹 브라우저에서 위에 나와 있는 URL(예: http://localhost:5173)로 이동합니다. 앱 예에서는 Amazon S3 버킷의 객체 파일 이름 목록을 보여줍니다.

정리

이 자습서를 진행하는 동안 생성한 리소스를 정리하려면 다음을 수행합니다.

- Amazon S3 콘솔에서, 생성한 객체와 버킷(예: getstartedbucket)을 삭제합니다.

- IAM 콘솔에서, 역할 이름(예: getStartedRole)을 삭제합니다.
- Amazon Cognito 콘솔에서, 자격 증명 풀 이름(예: getStartedPool)을 삭제합니다.

다음에 대해 SDK를 설정합니다. JavaScript

이 섹션의 항목에서는 SDK에서 지원하는 웹 서비스에 액세스할 수 JavaScript 있도록 SDK를 설치하고 로드하는 방법을 설명합니다.

> **Note**
>
> React Native 개발자는 새 프로젝트를 만들 AWS Amplify 때 사용해야 합니다. AWS자세한 내용은 aws-sdk-react-native아카이브를 참조하십시오.

주제

필수 조건

Node.js를 아직 설치하지 않은 경우 서버에 설치합니다.

주제

AWS Node.js 환경 설정하기

애플리케이션을 실행할 수 있는 AWS Node.js 환경을 설정하려면 다음 방법 중 하나를 사용하십시오.

- Node.js가 사전 설치된 Amazon Machine Image(AMI)를 선택합니다. 그런 다음, 해당 AMI를 사용하여 Amazon EC2 인스턴스를 생성합니다. Amazon EC2 인스턴스를 생성할 때 AWS Marketplace에서 AMI를 선택합니다. 에서 Node.js AWS Marketplace 를 검색하고 Node.js (32비트 또는 64비트)의 사전 설치된 버전이 포함된 AMI 옵션을 선택합니다.
- Amazon EC2 인스턴스를 생성하고 해당 인스턴스에 Node.js를 설치합니다. Amazon Linux 인스턴스에 Node.js를 설치하는 방법에 관한 자세한 내용은 Amazon EC2 인스턴스에서 Node.js 설정 섹션을 참조하세요.

- Node.js 를 Lambda 함수로 실행하는 AWS Lambda 데 사용하여 서버리스 환경을 생성합니다. Lambda 함수 내에서 Node.js를 사용하는 방법에 관한 자세한 내용은 AWS Lambda 개발자 안내서의 프로그래밍 모델(Node.js) 단원을 참조하세요.
- Node.js 애플리케이션을 에 배포하십시오. AWS Elastic Beanstalk Elastic Beanstalk에서 Node.js를 사용하는 방법에 관한 자세한 내용은 AWS Elastic Beanstalk 개발자 안내서의 AWS Elastic Beanstalk에 Node.js 애플리케이션 배포 단원을 참조하세요.
- 를 사용하여 Node.js 애플리케이션 서버를 생성합니다 AWS OpsWorks. Node.js 사용에 대한 자세한 내용은 사용 AWS OpsWorks 설명서의 첫 Node.js 스택 만들기를 참조하십시오. AWS OpsWorks

지원되는 웹 브라우저

AWS SDK for JavaScript 는 모든 최신 웹 브라우저를 지원합니다.

버전 3.183.0 이상에서 SDK는 다음과 같은 최소 버전을 지원하는 ES2020 아티팩트를 JavaScript 사용합니다.

브라우저	버전
Google Chrome	80.0+
Mozilla Firefox	80.0+
Opera	63.0+
Microsoft Edge	80.0+
Apple Safari	14.1+
삼성 인터넷	12.0+

버전 3.182.0 이하에서 SDK는 다음과 같은 최소 버전을 지원하는 ES5 아티팩트를 JavaScript 사용합니다.

브라우저	버전
Google Chrome	49.0+

브라우저	버전
Mozilla Firefox	45.0+
Opera	36.0+
Microsoft Edge	12.0+
Windows Internet Explorer	N/A
Apple Safari	9.0+
Android 브라우저	76.0+
UC 브라우저	12.12+
삼성 인터넷	5.0+

> **Note**
>
> 와 같은 프레임워크는 SDK와 동일한 브라우저 지원을 제공하지 AWS Amplify 않을 수 있습니다. JavaScript 자세한 내용은 AWS Amplify 설명서를 참조하세요.

다음에 대한 SDK를 설치하세요. JavaScript

모든 서비스를 SDK 또는 모든 AWS 지역에서 즉시 사용할 수 있는 것은 아닙니다.

Node.js 패키지 관리자를 AWS SDK for JavaScript 사용하는 npm에서 서비스를 설치하려면 명령 프롬프트에 다음 명령을 입력합니다. 여기서 SERVICE는 #### 이름 (예:) 입니다. s3

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

AWS SDK for JavaScript 서비스 클라이언트 패키지의 전체 목록은 AWS SDK for JavaScript API 참조 안내서를 참조하십시오.

에 대한 SDK를 로드하십시오. JavaScript

SDK를 설치한 후 `import`를 사용하여 노드 애플리케이션에 클라이언트 패키지를 로드할 수 있습니다. 예를 들어, Amazon S3 클라이언트와 Amazon S3 ListBuckets명령을 로드하려면 다음을 사용하십시오.

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

다음에 대해 SDK를 구성합니다. JavaScript

SDK for를 사용하여 API를 사용하여 웹 서비스를 JavaScript 호출하려면 먼저 SDK를 구성해야 합니다. 최소한 다음을 구성해야 합니다.

- AWS 서비스를 요청할 지역
- 코드 인증 방법 AWS

이러한 설정 외에도 AWS 리소스에 대한 권한도 구성해야 할 수 있습니다. 예를 들어 Amazon S3 버킷에 대한 액세스를 제한하거나 읽기 전용 액세스만 가능하도록 Amazon DynamoDB 테이블을 제한할 수 있습니다.

AWS SDK 및 도구 참조 안내서에는 많은 SDK에 공통적인 설정, 기능 및 기타 기본 개념도 포함되어 있습니다. AWS

이 섹션의 항목에서는 Node.js JavaScript SDK를 구성하고 웹 브라우저에서 JavaScript 실행하는 방법을 설명합니다.

주제

서비스별 구성

서비스 객체에 구성 정보를 전달하여 SDK를 구성할 수 있습니다.

서비스 수준 구성은 개별 서비스에 대한 중요한 제어를 제공하므로 요구 사항이 기본 구성과 다를 때 개별 서비스 객체의 구성을 업데이트할 수 있습니다.

> **Note**
>
> 버전 2.x에서는 AWS SDK for JavaScript 서비스 구성을 개별 클라이언트 생성자에 전달할 수 있었습니다. 그러나 이러한 구성은 먼저 글로벌 SDK 구성 `AWS.config`의 복사본에 자동으로 병합됩니다.

또한 `AWS.config.update({/* params */})`를 직접적으로 호출하면 기존 클라이언트가 아니라 업데이트 호출이 이루어진 후 인스턴스화된 서비스 클라이언트에 대한 구성만 업데이트됩니다.
이 동작은 빈번하게 혼란을 야기했으며, 이로 인해 순방향 호환 방식으로 서비스 클라이언트의 하위 집합에만 영향을 주는 구성을 글로벌 객체에 추가하기가 어려워졌습니다. 버전 3에서는 더 이상 SDK로 관리되는 글로벌 구성이 없습니다. 인스턴스화된 각 서비스 클라이언트에 구성을 전달해야 합니다. 여전히 동일한 구성을 여러 클라이언트에서 공유할 수는 있지만, 해당 구성이 글로벌 상태와 자동으로 병합되지는 않습니다.

서비스별 구성 설정

SDK에서 사용하는 각 서비스는 해당 서비스 API의 일부인 서비스 객체를 통해 액세스됩니다. JavaScript 예를 들어, Amazon S3 서비스에 액세스하려면 Amazon S3 서비스 객체를 생성합니다. 해당 서비스 객체에 대한 생성자의 일부인 서비스별 구성 설정을 지정할 수 있습니다.

예를 들어 여러 AWS 지역의 Amazon EC2 객체에 액세스해야 하는 경우 각 지역에 대한 Amazon EC2 서비스 객체를 생성한 다음 이에 따라 각 서비스 객체의 지역 구성을 설정하십시오.

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

지역을 설정합니다. AWS

AWS 지역은 같은 지리적 영역에 있는 이름이 지정된 AWS 리소스 집합입니다. 리전의 한 가지 예로 미국 동부(버지니아 북부) 리전인 `us-east-1`을 들 수 있습니다. SDK에서 서비스 클라이언트를 만들 때 지역을 지정하면 JavaScript SDK가 해당 지역의 서비스에 액세스할 수 있습니다. 일부 서비스는 특정 리전에서만 사용할 수 있습니다.

용 SDK는 기본적으로 지역을 선택하지 JavaScript 않습니다. 하지만 환경 변수나 공유 구성 `config` 파일을 사용하여 AWS 지역을 설정할 수 있습니다.

클라이언트 클래스 생성자에서

서비스 객체를 인스턴스화할 때 다음과 같이 해당 리소스의 AWS 지역을 클라이언트 클래스 생성자의 일부로 지정할 수 있습니다.

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

환경 변수를 사용하세요.

AWS_REGION 환경 변수를 사용하여 리전을 설정할 수 있습니다. 이 변수를 정의하면 for의 SDK가 이 변수를 JavaScript 읽고 사용합니다.

공유 구성 파일 사용

공유 자격 증명 파일을 통해 SDK에서 사용할 자격 증명을 저장할 수 있는 것처럼 AWS 지역 및 기타 구성 설정을 SDK에서 사용할 이름이 지정된 config 공유 파일에 보관할 수 있습니다. AWS_SDK_LOAD_CONFIG환경 변수가 올바른 값으로 설정된 경우 SDK는 파일이 로드될 때 config 파일을 JavaScript 자동으로 검색합니다. config 파일을 저장하는 위치는 운영 체제에 따라 다릅니다.

- Linux, macOS 또는 Unix 사용자 – ~/.aws/config
- Windows 사용자 - C:\Users\USER_NAME\.aws\config

아직 공유 config 파일이 없는 경우, 지정된 디렉터리에 하나를 생성할 수 있습니다. 다음 예제의 경우 config 파일에서 리전과 출력 형식을 둘 다 설정합니다.

```
[default]
   region=us-west-2
   output=json
```

공유 config 및 credentials 파일 사용에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

리전 설정을 위한 우선순위

리전 설정의 우선순위는 다음과 같습니다.

1. 어떤 리전이 클라이언트 클래스 생성자로 전달된 경우 이 리전이 사용됩니다.
2. 환경 변수에 리전을 설정한 경우 이 리전이 사용됩니다.
3. 그렇지 않으면 공유 구성 파일에 정의된 리전이 사용됩니다.

자격 증명 설정

AWS 자격 증명을 사용하여 서비스를 호출하는 사람과 요청된 리소스에 대한 액세스가 허용되는지 여부를 식별합니다.

웹 브라우저에서 실행하든 Node.js 서버에서 실행하든, JavaScript 코드가 API를 통해 서비스에 액세스하려면 먼저 유효한 자격 증명을 얻어야 합니다. 보안 인증을 서비스 객체에 직접 전달함으로써 서비스별로 보안 인증을 설정할 수 있습니다.

Node.js 버전과 웹 브라우저 간에는 자격 증명을 설정하는 방법이 여러 가지가 있습니다. JavaScript 이 섹션의 주제에서는 Node.js 또는 웹 브라우저에서 인증 자격 증명을 설정하는 방법을 설명합니다. 각각의 경우에 옵션은 권장 순서로 제공됩니다.

보안 인증에 대한 모범 사례

인증 자격 증명을 올바르게 설정하면 애플리케이션 또는 브라우저 스크립트가 필요한 서비스 및 리소스에 액세스할 수 있는 동시에 미션 크리티컬 애플리케이션에 미치거나 중요한 데이터를 손상시킬 수 있는 보안 문제에 대한 노출을 최소화할 수 있습니다.

인증 자격 증명을 설정할 때 적용되는 중요한 원칙은 항상 작업에 필요한 최소 권한을 부여하는 것입니다. 최소 권한을 초과하는 권한을 제공하고 그 결과로 추후 발견할 수 있는 보안 문제를 해결하기 보다는, 리소스에 대한 최소 권한을 제공하고 필요에 따라 권한을 추가하는 것이 더 안전합니다. 예를 들어 Amazon S3 버킷 또는 DynamoDB 테이블의 객체 같은 개별 리소스를 읽고 쓸 필요가 있지 않은 한, 그러한 권한을 읽기 전용으로 설정합니다.

최소 권한 부여에 관한 자세한 내용은 IAM 사용 설명서의 모범 사례 주제에서 최소 권한 적용 단원을 참조하세요.

주제

- Node.js 에서 자격 증명 설정
- 웹 브라우저에서 자격 증명 설정

Node.js 에서 자격 증명 설정

현지에서 개발 중이고 고용주로부터 인증 방법을 알려주지 않은 신규 사용자를 등록하는 것이 좋습니다 AWS IAM Identity Center. 자세한 설명은 AWS를 사용한 SDK 인증 섹션을 참조하세요.

Node.js에서 SDK에 인증 자격 증명을 제공하는 방법에는 여러 가지가 있습니다. 그 가운데는 더 안전한 방법도 있고 애플리케이션 개발 시에 더 편리한 방법도 있습니다. Node.js에서 보안 인증을 얻을 때 로드하는 JSON 파일, 환경 변수 등 둘 이상의 소스를 사용하는 경우 주의해야 합니다. 변경 발생에 대한 인식 없이 코드가 실행되는 권한을 변경할 수 있습니다.

AWS SDK for JavaScript V3에서는 Node.js 파일에 기본 자격 증명 공급자 체인을 제공하므로 자격 증명 공급자를 명시적으로 제공할 필요는 없습니다. 기본 보안 인증 공급자 체인은 보안 인증이 소스 중

하나에서 반환될 때까지 지정된 우선순위에 따라 여러 가지 다양한 소스의 보안 인증을 확인하려고 시도합니다. V3용 SDK용 자격 증명 공급자 체인은 여기에서 찾을 수 있습니다. JavaScript

보안 인증 공급자 체인

모든 SDK에는 AWS 서비스에 요청하는 데 사용할 유효한 보안 인증을 얻기 위해 확인하는 일련의 장소(또는 소스)가 있습니다. 유효한 보안 인증 정보를 찾은 후에는 검색이 중지됩니다. 이러한 체계적인 검색을 기본 보안 인증 공급자 체인이라고 합니다.

체인의 각 단계마다 값을 설정하는 다양한 방법이 있습니다. 코드에서 값을 직접 설정하는 것이 항상 우선하며, 환경 변수로 설정, 공유 파일에서 설정 순입니다. AWS `config` 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Precedence of settings 단원을 참조하세요.

AWS SDK 및 도구 참조 안내서에는 모든 AWS SDK 및 에서 사용하는 SDK 구성 설정에 대한 정보가 있습니다. AWS CLI공유 파일을 통해 SDK를 구성하는 방법에 대한 자세한 내용은 공유 구성 및 AWS `config` 자격 증명 파일을 참조하십시오. 환경 변수 설정을 통해 SDK를 구성하는 방법에 관해 자세히 알아보려면 Environment variables support 단원을 참조하세요.

인증을 위해 는 다음 AWS SDK for JavaScript 표에 나열된 순서대로 자격 증명 공급자를 확인합니다. AWS

AWS SDK for JavaScript API 참조 자격 증명 공급자 방법 (우선 순위 기준)	사용 가능한 보안 인증 공급자	AWS SDK 및 도구 참조 가이드
`fromEnv()`	AWS 환경 변수의 액세스 키	AWS 액세스 키
`fromSSO()`	AWS IAM Identity Center. 이 설명서의 AWS를 사용한 SDK 인증 단원을 참조하세요.	IAM Identity Center 보안 인증 공급자
`fromIni()`	AWS 공유 `config` 및 `credentials` 파일의 액세스 키	AWS 액세스 키
	신뢰할 수 있는 엔터티 공급자 (예:`AWS_ROLE_ARN`)	IAM 역할 수임
	AWS Security Token Service (AWS STS) 의 웹 ID 토큰	웹 자격 증명 또는 OpenID Connect와 페더레이션

AWS SDK for JavaScript API 참조 자격 증명 공급자 방법 (우선 순위 기준)	사용 가능한 보안 인증 공급자	AWS SDK 및 도구 참조 가이드
	Amazon Elastic Container Service(Amazon ECS) 보안 인증	컨테이너 보안 인증 공급자
	Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스 프로파일 보안 인증(IMDS 보안 인증 공급자)	IMDS 보안 인증 공급자
	프로세스 보안 인증 공급자	프로세스 보안 인증 공급자
	AWS IAM Identity Center 자격 증명	IAM Identity Center 보안 인증 공급자
`fromProcess()`	프로세스 보안 인증 공급자	프로세스 보안 인증 공급자
`fromTokenFile()`	AWS Security Token Service (AWS STS) 의 웹 ID 토큰	웹 자격 증명 또는 OpenID Connect와 페더레이션
`fromContainerMetadata()`	Amazon Elastic Container Service(Amazon ECS) 보안 인증	컨테이너 보안 인증 공급자
`fromInstanceMetadata()`	Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스 프로파일 보안 인증(IMDS 보안 인증 공급자)	IMDS 보안 인증 공급자

신규 사용자에게 권장되는 시작하기 접근 방식을 따랐다면 시작하기 항목의 AWS를 사용한 SDK 인증 중에 AWS IAM Identity Center 인증을 설정합니다. 상황에 따라 다른 인증 방법이 유용할 수 있습니다. 보안 위험을 방지하려면 항상 단기 보안 인증을 사용하는 것이 좋습니다. 다른 인증 방법 절차에 대해서는 AWS SDK 및 도구 참조 가이드의 Authentication and access 단원을 참조하세요.

이 섹션의 주제에서는 인증 자격 증명을 Node.js로 로드하는 방법을 설명합니다.

주제

- Amazon EC2의 IAM 역할에서 Node.js 에 자격 증명을 로드합니다.
- Node.js Lambda 함수에 대한 자격 증명을 로드합니다.

Amazon EC2의 IAM 역할에서 Node.js 에 자격 증명을 로드합니다.

Amazon EC2 인스턴스에서 Node.js 애플리케이션을 실행하는 경우 Amazon EC2의 IAM 역할을 활용하여 해당 인스턴스에 자격 증명을 자동으로 제공할 수 있습니다. IAM 역할을 사용하도록 인스턴스를 구성하면 SDK가 애플리케이션의 IAM 보안 인증을 자동으로 선택하므로 보안 인증을 수동으로 제공할 필요가 없습니다.

Amazon EC2 인스턴스에 IAM 역할을 추가하는 방법에 관한 자세한 내용은 Amazon EC2의 IAM 역할 단원을 참조하세요.

Node.js Lambda 함수에 대한 자격 증명을 로드합니다.

함수를 생성할 때는 AWS Lambda 함수를 실행할 권한이 있는 특수 IAM 역할을 생성해야 합니다. 이 역할을 실행 역할이라고 합니다. Lambda 함수를 설정할 때 해당 실행 역할로 생성한 IAM 역할을 지정해야 합니다.

실행 역할은 Lambda 함수에 다른 웹 서비스를 실행 및 간접 호출하는 데 필요한 보안 인증을 제공합니다. 따라서 Lambda 함수 내에 쓰는 Node.js 코드에 보안 인증을 제공할 필요가 없습니다.

Lambda 실행 역할 생성에 관한 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 리소스 액세스 권한 단원을 참조하세요.

웹 브라우저에서 자격 증명 설정

브라우저 스크립트에서 SDK에 인증 자격 증명을 제공하는 방법에는 여러 가지가 있습니다. 그 가운데는 더 안전한 방법도 있고 스크립트 개발 시에 더 편리한 방법도 있습니다.

다음은 권장 순서로 보안 인증을 제공할 수 있는 방법입니다.

1. Amazon Cognito 자격 증명을 사용하여 사용자를 인증하고 보안 인증 제공
2. 웹 연동 자격 증명 사용

> **Warning**
>
> 스크립트에 AWS 자격 증명을 하드 코딩하지 않는 것이 좋습니다. 인증 자격 증명을 하드 코딩하면 액세스 키 ID 및 보안 액세스 키가 노출될 위험이 있습니다.

주제

- Amazon Cognito 자격 증명을 사용하여 사용자를 인증합니다.

Amazon Cognito 자격 증명을 사용하여 사용자를 인증합니다.

브라우저 스크립트용 AWS 자격 증명을 얻는 권장 방법은 Amazon Cognito ID 자격 증명 클라이언트를 사용하는 것입니다. `CognitoIdentityClient` Amazon Cognito를 사용하면 타사 자격 증명 공급자를 통해 사용자를 인증할 수 있습니다.

Amazon Cognito 자격 증명을 사용하려면 먼저, Amazon Cognito 콘솔에서 자격 증명 풀을 생성해야 합니다. 자격 증명 풀은 애플리케이션이 사용자에게 제공하는 자격 증명 그룹을 나타냅니다. 사용자에게 제공되는 자격 증명은 각 사용자 계정을 고유하게 식별합니다. Amazon Cognito 자격 증명(identity)은 자격 증명(credential)이 아닙니다. AWS Security Token Service ()AWS STS의 웹 자격 증명 연동 지원을 사용하여 자격 증명으로 교환됩니다.

Amazon Cognito에서는 여러 자격 증명 공급자의 자격 증명 추상화를 관리할 수 있습니다. 그러면 로드되는 자격 증명이 AWS STS의 인증 자격 증명과 교환됩니다.

Amazon Cognito 자격 증명 자격 증명 객체를 구성합니다.

아직 자격 증명 풀을 생성하지 않은 경우 Amazon Cognito 클라이언트를 구성하기 전에 Amazon Cognito 콘솔에서 브라우저 스크립트와 함께 사용할 자격 증명 풀을 생성합니다. 자격 증명 풀에 대한 인증 IAM 역할과 미인증 IAM 역할을 모두 생성하고 연결합니다. 자세한 내용은 Amazon Cognito 개발자 안내서의 자습서: 자격 증명 풀 생성 단원을 참조하세요.

미인증 사용자는 자격 증명이 인증되지 않았으므로 이 역할은 앱의 게스트 사용자에게 적합하거나 사용자의 자격 증명 인증 여부가 중요하지 않은 경우에 적합합니다. 인증받은 사용자는 자격 증명을 확인하는 타사 자격 증명 공급자를 통해 애플리케이션에 로그인합니다. 미인증 사용자의 액세스 권한을 허용하지 않도록 리소스 권한 범위를 충분히 정했는지 확인하세요.

자격 증명 풀을 구성한 후 `@aws-sdk/credential-providers`에서 `fromCognitoIdentityPool` 메서드를 사용하여 자격 증명 풀에서 보안 인증을 검색합니다. Amazon S3 클라이언트를 생성하는 다음 예에서는 *`AWS_REGION`*을 리전으로 바꾸고 *`IDENTITY_POOL_ID`*를 자격 증명 풀 ID로 바꿉니다.

```
// 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.
        },
  })
});
```

선택 사항인 `logins` 속성은 공급자의 자격 증명 토큰에 대한 자격 증명 공급자 이름의 맵입니다. 자격 증명 공급자에게서 토큰을 받는 방법은 어떤 공급자를 사용하느냐에 따라 다릅니다. 예를 들어 Amazon Cognito 사용자 풀을 인증 공급자로 사용하는 경우 아래와 비슷한 메서드를 사용할 수 있습니다.

```
// 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;
};
```

인증되지 않은 사용자를 인증된 사용자로 전환

Amazon Cognito는 인증된 사용자와 인증되지 않은 사용자를 모두 지원합니다. 인증되지 않은 사용자는 자격 증명 공급자로 로그인하지 않았더라도 리소스에 대한 액세스 권한을 받습니다. 이 액세스 권한 등급은 로그인하기 전에 사용자에게 콘텐츠를 표시하는 데 유용합니다. 각 미인증 사용자는 개별적으로 로그인되지 않고 인증되지 않았더라도 Amazon Cognito에 고유한 자격 증명이 있습니다.

처음에 인증되지 않은 사용자

사용자는 일반적으로 `logins` 속성 없이 구성 객체의 인증 자격 증명 속성을 설정한 인증되지 않은 역할로 시작합니다. 이 경우 기본 보안 인증은 다음과 같을 수 있습니다.

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

인증된 사용자로 전환

인증되지 않은 사용자가 자격 증명 공급자에 로그인한 상태에서 현재 사용자가 토큰을 갖고 있다면, 보안 인증 객체를 업데이트하고 `logins` 토큰을 추가하는 사용자 지정 함수를 호출하여 인증되지 않은 사용자를 인증된 사용자로 전환할 수 있습니다.

```
// 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;
```

```
}
```

Node.js 고려 사항

Node.js 코드가 그렇긴 하지만 Node.js AWS SDK for JavaScript 에서 를 사용하는 것은 JavaScript 브라우저 스크립트에서 SDK를 사용하는 것과 다를 수 있습니다. 일부 API 메서드는 Node.js에서 작동하지만 브라우저 스크립트에서는 작동하지 않습니다. 마찬가지로 브라우저 스크립트에서는 작동하지만 Node.js에서 작동하지 않는 API 메서드도 있습니다. 또한 일부 API를 성공적으로 사용할 수 있는지는 `File System (fs)` 모듈 등의 여러 Node.js 모듈을 가져와 사용하는 것과 같은 일반적인 Node.js 코딩 패턴에 익숙한지 여부에 달려 있습니다.

내장된 Node.js 모듈 사용

Node.js는 설치하지 않고도 사용할 수 있는 기본 제공 모듈 모음을 제공합니다. 이러한 모듈을 사용하려면 `require` 메서드로 객체를 생성하여 모듈 이름을 지정합니다. 예를 들어 기본 제공 HTTP 모듈을 포함시키려면 다음을 사용합니다.

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

모듈의 메서드가 해당 객체의 메서드인 것처럼 모듈의 메서드를 호출합니다. 예를 들어 다음은 HTML 파일을 읽는 코드입니다.

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

Node.js가 제공하는 모든 기본 제공 모듈의 전체 목록은 Node.js 웹 사이트의 Node.js documentation을 참조하세요.

npm 패키지 사용

기본 제공 모듈 외에도 Node.js 패키지 관리자인 npm의 타사 코드를 포함 및 통합할 수 있습니다. 이는 오픈 소스 Node.js 패키지와 이러한 패키지를 설치하기 위한 명령줄 인터페이스의 리포지토리입니다.

npm과 현재 사용 가능한 패키지 목록에 대한 자세한 내용은 https://www.npmjs.com을 참조하세요. 여기에서 사용할 수 있는 추가 Node.js 패키지에 대해서도 알아볼 수 있습니다. GitHub

Node.js 에서 MaxSockets 설정하기

Node.js에서 오리진당 최대 연결 수를 설정할 수 있습니다. `maxSockets`이 설정된 경우, 하위 HTTP 클라이언트가 요청을 대기열에 넣고 소켓이 사용 가능해지면 소켓에 요청을 할당합니다.

그러면 지정된 오리진에 한 번에 동시 요청할 수 있는 수의 상한을 설정할 수 있습니다. 이 값을 낮추면 수신하는 조절 또는 시간 초과 오류 수를 줄일 수 있습니다. 그러나 소켓이 사용 가능해질 때까지 요청이 대기되므로 메모리 사용량도 증가할 수 있습니다.

다음 예는 DynamoDB 클라이언트에 대해 `maxSockets`를 설정하는 방법을 보여줍니다.

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

값이나 객체를 JavaScript 제공하지 않는 경우 SDK의 `maxSockets` 값은 50입니다. `Agent` `Agent`객체를 제공하면 해당 `maxSockets` 값이 사용됩니다. Node.js 설정에 `maxSockets` 대한 자세한 내용은 Node.js 설명서를 참조하십시오.

v3.521.0부터 다음과 같은 AWS SDK for JavaScript간단한 구문을 사용하여 구성할 수 있습니다. `requestHandler`

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

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

```
});
```

Node.js 내에서 keep-alive를 사용하여 연결을 재사용하십시오.

기본 Node.js HTTP/HTTPS 에이전트는 모든 새 요청에 대해 새로운 TCP 연결을 생성합니다. 새 연결을 설정하는 데 드는 비용을 방지하기 위해 SDK용 SDK는 TCP 연결을 재사용합니다. JavaScript

Amazon DynamoDB 쿼리와 같은 수명이 짧은 작업의 경우 TCP 연결 설정에 따른 지연 시간 오버헤드가 작업 자체보다 클 수 있습니다. 또한 DynamoDB 유휴 암호화가 AWS KMS와 통합되어 있기 때문에 데이터베이스에서 각 작업마다 AWS KMS 새 캐시 항목을 다시 설정해야 하는 지연 시간이 발생할 수 있습니다.

또한 DynamoDB 클라이언트에 대한 다음 예와 같이 서비스별 클라이언트 단위로 이러한 연결을 유지하는 것을 비활성화할 수도 있습니다.

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

`keepAlive`가 활성화된 경우 기본값이 1000ms인 `keepAliveMsecs`를 사용하여 TCP 연결 유지 패킷에 대한 초기 지연을 설정할 수도 있습니다. 자세한 내용은 Node.js 설명서를 참조하세요.

Node.js 프록시를 구성하십시오.

인터넷에 직접 연결할 수 없는 경우 SDK용 SDK는 타사 HTTP 에이전트를 통한 HTTP 또는 HTTPS 프록시 사용을 JavaScript 지원합니다.

타사 HTTP 에이전트를 찾으려면 npm에서 "HTTP proxy"를 검색하세요.

타사 HTTP 에이전트 프록시를 설치하려면 명령 프롬프트에 다음을 입력합니다. 여기서 *PROXY*는 npm 패키지의 이름입니다.

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

애플리케이션에서 프록시를 사용하려면 DynamoDB 클라이언트에 대한 다음 예와 같이 `httpAgent` 및 `httpsAgent` 속성을 사용합니다.

```
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`는 `httpsAgent`와 동일하지 않으며 클라이언트에서 대부분의 직접 호출이 `https`로 이루어지므로 둘 다 설정해야 합니다.

Node.js 파일에 인증서 번들 등록

Node.js용 기본 트러스트 스토어에는 AWS 서비스에 액세스하는 데 필요한 인증서가 포함되어 있습니다. 일부 경우에는 특정 인증서 집합만 포함하는 것이 좋을 수도 있습니다.

이 예제에서는 지정된 인증서가 제공되지 않은 경우 디스크의 특정 인증서를 사용하여 연결을 거부하는 `https.Agent`를 생성합니다. 새로 생성한 `https.Agent`는 DynamoDB 클라이언트에서 사용됩니다.

```
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
  })
});
```

브라우저 스크립트 고려 사항

다음 항목에서는 브라우저 내 스크립트 사용에 대한 특별 고려 사항을 설명합니다. AWS SDK for JavaScript

주제

브라우저용 SDK 구축

JavaScript 버전 2용 SDK (V2) 와 달리 V3는 기본 서비스 세트에 대한 지원이 포함된 JavaScript 파일로 제공되지 않습니다. 대신 V3에서는 필요한 JavaScript 파일용 SDK만 번들로 제공하고 브라우저에 포함할 수 있으므로 오버헤드가 줄어듭니다. Webpack을 사용하여 JavaScript 파일에 필요한 SDK와 필요한 추가 타사 패키지를 단일 `Javascript` 파일로 묶고 태그를 사용하여 브라우저 스크립트에 로드하는 것이 좋습니다. `<script>` Webpack에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요. Webpack을 사용하여 브라우저에 V3 SDK를 로드하는 예제는 을 참조하십시오 JavaScript . DynamoDB에 데이터를 제출하는 앱 빌드

브라우저에서 CORS를 적용하는 환경 외부에서 SDK를 사용하고 SDK에서 제공하는 모든 서비스에 액세스하려는 경우 리포지토리를 복제하고 SDK의 기본 호스팅 버전을 빌드하는 것과 동일한 빌드 도구를 실행하여 로컬에서 SDK의 사용자 지정 사본을 만들 수 있습니다. JavaScript 다음 섹션에서는 추가 서비스 및 API 버전으로 SDK를 빌드하는 단계를 설명합니다.

SDK 빌더를 사용하여 SDK를 빌드할 수 있습니다. JavaScript

> **Note**
>
> Amazon Web Services 버전 3(V3)는 더 이상 브라우저 빌더를 지원하지 않습니다. 브라우저 애플리케이션의 대역폭 사용량을 최소화하려면 명명된 모듈을 가져와서 번들링하여 크기를 줄이는 것이 좋습니다. 번들링에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

교차 오리진 리소스 공유(CORS)

CORS(Cross-origin 리소스 공유)는 최신 웹 브라우저의 보안 기능입니다. 이 기능을 사용하면 웹 브라우저가 어떤 도메인이 외부 웹 사이트 또는 서비스를 요청할 수 있을지 협상할 수 있습니다.

대부분의 리소스 요청이 외부 도메인(예: 웹 서비스용 엔드포인트)으로 전송되기 때문에 AWS SDK for JavaScript 를 사용해 브라우저 애플리케이션을 개발하는 경우 CORS는 중요한 고려 대상입니다. JavaScript 환경에 CORS 보안이 적용되는 경우 서비스와 함께 CORS를 구성해야 합니다.

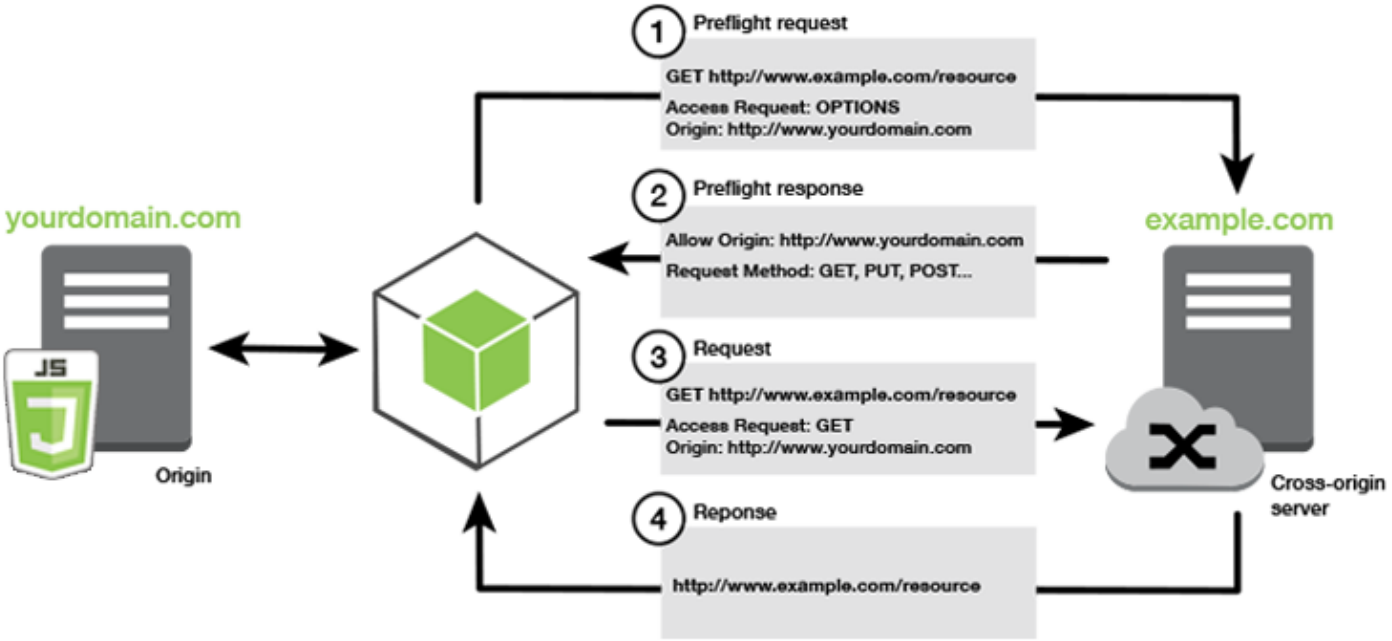
CORS는 다음을 기반으로 교차 오리진 요청 시 리소스 공유 허용 여부를 결정합니다.

- 요청을 수행한 특정 도메인
- 수행 중인 HTTP 요청 유형(GET, PUT, POST, DELETE 등)

CORS 작동 방식

가장 간단한 경우 브라우저 스크립트는 다른 도메인 내 서버의 리소스에 대해 GET 요청을 수행합니다. 요청이 GET 요청을 제출하도록 승인된 도메인에서 전송된 경우 해당 서버의 CORS 구성에 따라 cross-origin 서버는 요청된 리소스를 반환하여 응답합니다.

요청하는 도메인 또는 HTTP 요청 유형이 승인되지 않은 경우에는 요청이 거부됩니다. 그러나 CORS는 요청을 실제로 제출하기 전에 요청을 사전에 보낼 수 있습니다. 이 경우 preflight 요청은 `OPTIONS` 액세스 요청 작업을 전송하는 방식으로 이루어집니다. cross-origin 서버의 CORS 구성이 요청하는 도메인에 대한 액세스 권한을 부여하는 경우 서버에서는 요청하는 도메인이 요청한 리소스에 대해 수행 가능한 HTTP 요청 유형을 모두 나열하는 preflight 응답으로 회신합니다.



CORS 구성이 필요한가요?

Amazon S3 버킷에서 작업을 수행하려면 먼저 해당 버킷에 CORS 구성이 필요합니다. 일부 JavaScript 환경에서는 CORS가 적용되지 않을 수 있으므로 CORS를 구성할 필요가 없습니다. 예를 들어 Amazon S3 버킷에서 애플리케이션을 호스팅하고 `*.s3.amazonaws.com` 또는 일부 다른 특정 엔드포인트에서 리소스에 액세스하는 경우에는 요청이 외부 도메인에 액세스하지 않습니다. 따라서 이 구성에는 CORS가 필요하지 않습니다. 이 경우에도 CORS는 Amazon S3 이외의 서비스에는 여전히 사용됩니다.

Amazon S3 버킷에 대한 CORS를 구성합니다.

Amazon S3 콘솔에서 CORS를 사용하도록 Amazon S3 버킷을 구성할 수 있습니다.

AWS 웹 서비스 관리 콘솔에서 CORS를 구성하는 경우 JSON을 사용하여 CORS 구성을 생성해야 합니다. 새 AWS 웹 서비스 관리 콘솔은 JSON CORS 구성만 지원합니다.

> **Important**
>
> 새 AWS 웹 서비스 관리 콘솔에서는 CORS 구성이 JSON이어야 합니다.

1. AWS 웹 서비스 관리 콘솔에서 Amazon S3 콘솔을 열고 구성하려는 버킷을 찾은 다음 해당 확인란을 선택합니다.

2. 열리는 창에서 권한을 선택합니다.
3. 권한 탭에서 CORS 구성을 선택합니다.
4. CORS 구성 편집기에 CORS 구성을 입력한 다음, 저장을 선택합니다.

CORS 구성은 <CORSRule> 내에 일련의 규칙이 포함된 XML 파일입니다. 구성에는 규칙이 최대 100개까지 있을 수 있습니다. 규칙은 다음 태그 중 하나로 정의합니다.

- <AllowedOrigin> - 교차 도메인 요청을 수행하도록 허용하는 도메인 오리진을 지정합니다.
- <AllowedMethod> - 교차 도메인 요청에서 허용하는 요청 유형(GET, PUT, POST, DELETE, HEAD)을 지정합니다.
- <AllowedHeader> - preflight 요청에서 허용되는 헤더를 지정합니다.

구성의 예는 Amazon Simple Storage Service 사용 설명서의 교차 오리진 리소스 공유(CORS) 사용 단원을 참조하세요.

CORS 구성의 예

다음 CORS 구성 예를 통해 사용자는 example.org 도메인에서 버킷 내부의 객체를 보거나 추가하거나 제거하거나 업데이트할 수 있습니다. 그러나 웹 사이트의 도메인으로 <AllowedOrigin> 범위를 지정하는 것이 좋습니다. 모든 도메인을 허용하려면 "*"를 지정할 수 있습니다.

> **Important**
>
> 새 S3 콘솔에서 CORS 구성은 JSON이어야 합니다.

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"]
    }
]
```

이 구성은 사용자가 버킷에 대해 작업을 수행하도록 허용하지 않고, 브라우저의 보안 모델이 Amazon S3에 대한 요청을 허용하도록 합니다. 권한은 버킷 권한 또는 IAM 역할 권한을 통해 구성해야 합니다.

ExposeHeader를 사용하면 SDK가 Amazon S3에서 반환된 응답 헤더를 읽도록 할 수 있습니다. 예를 들어 PUT 또는 멀티파트 업로드에서 ETag 헤더를 읽으려면 이전 예와 같이 구성에 ExposeHeader 태그를 포함해야 합니다. SDK는 CORS 구성을 통해 노출된 헤더에만 액세스할 수 있습니다. 객체에 대해 메타데이터를 설정한 경우 값은 접두사 x-amz-meta-가 붙은 헤더(예: x-amz-meta-my-custom-header)로 반환되며 동일한 방식으로 노출되어야 합니다.

웹팩이 포함된 번들 애플리케이션

브라우저 스크립트 또는 Node.js에서 웹 애플리케이션이 코드 모듈을 사용하면 종속성이 생성됩니다. 이러한 코드 모듈에는 자체 종속성이 있을 수 있어 애플리케이션 작동에 필요한 상호 연결된 모듈 모음이 생깁니다. 종속성을 관리하려면 webpack과 같은 모듈 번들러를 사용하면 됩니다.

webpack 모듈 번들러는 애플리케이션 코드를 구문 분석하여 import 또는 require 문을 검색해 애플리케이션에 필요한 모든 자산이 포함된 번들을 생성합니다. 이렇게 하면 웹 페이지를 통해 자산을 쉽게 제공할 수 있습니다. 용 SDK는 출력 번들에 포함할 종속성 중 webpack 하나로 포함될 JavaScript 수 있습니다.

에 대한 webpack 자세한 내용은 의 webpack 모듈 번들러를 참조하십시오. GitHub

웹팩 설치

webpack 모듈 번들러를 설치하려면 먼저, Node.js 패키지 관리자인 npm이 설치되어 있어야 합니다. 다음 명령을 입력하여 webpack CLI와 JavaScript 모듈을 설치합니다.

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

webpack과 함께 자동으로 설치되는 파일 및 디렉터리 경로 작업을 위해 path 모듈을 사용하려면 Node.js path-browserify 패키지를 설치해야 할 수도 있습니다.

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

웹팩을 구성합니다.

기본적으로 Webpack은 프로젝트의 루트 webpack.config.js 디렉터리에 이름이 지정된 JavaScript 파일을 검색합니다. 이 파일은 구성 옵션을 지정합니다. 다음은 WebPack 버전 5.0.0 webpack.config.js 이상의 구성 파일 예제입니다.

> **Note**
>
> Webpack 구성 요구 사항은 설치하는 Webpack 버전에 따라 다릅니다. 자세한 내용은 Webpack documentation을 참조하세요.

```
// 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"}]
  }
  **/
};
```

이 예에서는 `browser.js`가 진입점으로 지정됩니다. 이 진입점은 `webpack`이 가져온 모듈 검색을 시작하는 데 사용하는 파일입니다. 출력의 파일 이름은 `bundle.js`로 지정합니다. 이 출력 파일에는 응용 프로그램을 실행하는 데 필요한 모든 내용이 포함됩니다. JavaScript 진입점에 지정된 코드가 SDK for JavaScript 등의 다른 모듈을 가져오거나 요구하는 경우 구성에서 해당 코드를 지정하지 않아도 해당 코드가 번들로 제공됩니다.

웹팩을 실행합니다.

`webpack`을 사용하는 애플리케이션을 빌드하려면 `package.json` 파일의 `scripts` 객체에 다음을 추가합니다.

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

다음은 `webpack` 추가를 보여주는 `package.json` 파일의 예입니다.

```
{
  "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"
  }
}
```

애플리케이션을 빌드하려면 다음 명령을 입력합니다.

```
npm run build
```

그러면 webpack 모듈 번들러가 프로젝트의 루트 디렉터리에 지정한 JavaScript 파일을 생성합니다.

웹팩 번들을 사용하세요.

브라우저 스크립트에서 번들을 사용하려면 다음 예와 같이 <script> 태그를 사용해 번들을 포함하면 됩니다.

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

Node.js 번들

webpack을 사용하면 구성에서 node를 대상으로 지정하여 Node.js에서 실행되는 번들을 생성할 수 있습니다.

```
target: "node"
```

이는 디스크 공간이 제한된 환경에서 Node.js 애플리케이션을 실행하는 경우 유용합니다. 다음은 출력 대상으로 Node.js가 지정된 webpack.config.js 구성의 예입니다.

```
// 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"}]
  }
  **/
};
```

SDK에서 AWS 서비스를 사용하여 작업하기 JavaScript

AWS SDK for JavaScript v3는 클라이언트 클래스 컬렉션을 통해 지원하는 서비스에 대한 액세스를 제공합니다. 이러한 클라이언트 클래스에서는 일반적으로 서비스 객체라고 부르는 서비스 인터페이스 객체를 생성합니다. 지원되는 각 AWS 서비스에는 서비스 기능 및 리소스 사용을 위한 저수준 API를 제공하는 하나 이상의 클라이언트 클래스가 있습니다. 예를 들어 Amazon DynamoDB API는 `DynamoDB` 클래스를 통해 사용할 수 있습니다.

SDK를 통해 노출되는 서비스는 요청-응답 패턴을 JavaScript 따라 호출 애플리케이션과 메시지를 교환합니다. 이 패턴에서 서비스를 호출하는 코드는 해당 서비스의 엔트포인트에 HTTP/HTTPS 요청을 제출합니다. 이 요청에는 호출 중인 특정 기능을 성공적으로 호출하는 데 필요한 파라미터가 포함되어 있습니다. 호출된 서비스는 다시 요청자에게 보낼 응답을 생성합니다. 이 응답에는 작업에 성공에 성공한 경우에는 데이터가, 작업에 실패한 경우에는 오류 정보가 포함됩니다.

AWS 서비스 호출에는 시도된 모든 재시도를 포함하여 서비스 객체에 대한 작업의 전체 요청 및 응답 수명 주기가 포함됩니다. 요청에는 0개 이상의 속성이 JSON 파라미터로 포함됩니다. 응답은 작업과 관련된 객체에 캡슐화되고 콜백 함수 또는 프라미스와 같은 여러 기법 중 하나를 통해 요청자에게 반환됩니다. JavaScript

주제

서비스 객체 생성 및 호출

JavaScript API는 사용 가능한 대부분의 AWS 서비스를 지원합니다. JavaScriptAPI의 각 서비스는 서비스가 지원하는 모든 API를 호출하는 데 사용하는 `send` 메서드를 클라이언트 클래스에 제공합니다. API의 서비스 클래스, 작업 및 매개변수에 대한 자세한 내용은 JavaScript API 참조를 참조하십시오.

Node.js에서 SDK를 사용하는 경우 `import`를 사용하여 애플리케이션에 필요한 각 서비스에 대한 SDK 패키지를 추가합니다. 이 패키지는 현재의 모든 서비스를 지원합니다. 다음 예에서는 `us-west-1` 리전에 Amazon S3 서비스 객체를 생성합니다.

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

서비스 객체 파라미터를 지정하십시오.

서비스 객체의 메서드를 호출할 때 API에서 요구하는 대로 파라미터를 JSON으로 전달합니다. 예를 들어 Amazon S3에서 지정된 버킷과 키의 객체를 가져오려면 에서 `GetObjectCommand` 메서드로 다음 파라미터를 `S3Client` 전달하십시오. JSON 파라미터 전달에 대한 자세한 내용은 JSON으로 작업하기 섹션을 참조하세요.

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

Amazon S3 파라미터에 대한 자세한 내용은 API 레퍼런스의 @aws -sdk/client-s3를 참조하십시오.

서비스를 비동기적으로 호출합니다.

SDK를 통해 수행한 모든 요청은 비동기식입니다. 브라우저 스크립트를 작성할 때 이 점을 염두에 두어야 합니다. JavaScript 웹 브라우저에서 실행하는 경우 일반적으로 실행 스레드가 하나뿐입니다. AWS 서비스를 비동기로 호출한 후에도 브라우저 스크립트는 계속 실행되며, 이 프로세스에서 해당 비동기 결과에 따라 달라지는 코드를 실행한 후 반환할 수 있습니다.

AWS 서비스에 대한 비동기 호출에는 데이터를 사용할 수 있기 전에 코드에서 데이터를 사용하려고 하지 않도록 해당 호출을 관리하는 작업도 포함됩니다. 이 섹션의 주제에서는 비동기식 호출 관리의 필요성과 비동기식 호출 관리에 사용할 수 있는 다양한 기법에 대해 자세히 다룹니다.

이러한 기법 중 하나를 사용하여 비동기 직접 호출을 관리할 수 있지만, 모든 새 코드에 async/await를 사용하는 것이 좋습니다.

async/await

이 기법은 V3의 기본 동작이므로 사용하는 것이 좋습니다.

promise

async/await를 지원하지 않는 브라우저에서 이 기법을 사용하세요.

callback

매우 간단한 경우를 제외하고는 콜백을 사용하지 마세요. 하지만 마이그레이션 시나리오에는 유용할 수 있습니다.

주제

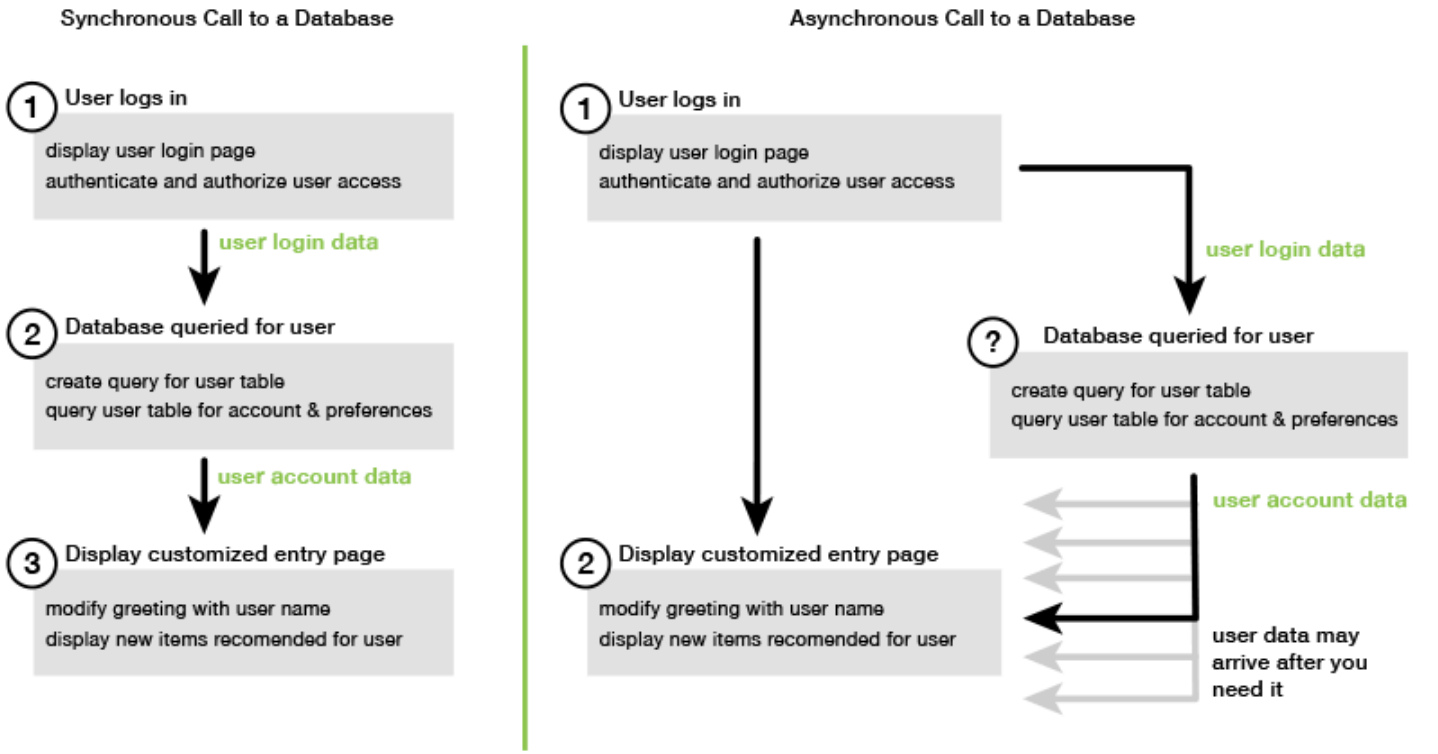
비동기 호출 관리

예를 들어, 전자 상거래 웹 사이트의 홈 페이지에서는 재방문 고객이 로그인할 수 있습니다. 로그인한 고객을 위한 혜택의 일부로 로그인 후 사이트에서는 고객의 특정 기본 설정에 맞춰 사이트를 맞춤화합니다. 사이트를 맞춤화하려면 다음을 수행해야 합니다.

1. 고객이 로그인하고 로그인 자격 증명으로 검증되어야 합니다.
2. 고객 데이터베이스에서 고객의 기본 설정이 요청됩니다.
3. 데이터베이스에서는 페이지 로드 전에 사이트를 맞춤화하는 데 사용되는 고객의 기본 설정을 제공합니다.

이러한 작업이 동기식으로 실행되면 다음 작업을 시작하기 전에 각 작업이 끝나야 합니다. 따라서 고객 기본 설정이 데이터베이스에서 반환될 때까지 웹 페이지 로딩을 완료할 수 없습니다. 그러나 데이터베이스 쿼리가 서버로 전송된 후 네트워크 병목 현상, 예외적으로 높은 데이터베이스 트래픽 또는 불안한 모바일 디바이스 연결 등으로 인해 고객 데이터 수신이 지연되거나 실패할 수 있습니다.

이러한 상황에서도 웹 사이트가 멈추지 않도록 하기 위해 데이터베이스를 비동기식으로 직접 호출합니다. 데이터베이스 호출을 실행한 후 비동기 요청을 보내면 코드가 계속해서 예상대로 실행됩니다. 비동기 호출의 응답을 적절하게 관리하지 못하면 데이터를 아직 사용할 수 없는데도 코드가 데이터베이스에서 다시 필요한 정보를 사용하려고 시도할 수 있습니다.



비동기/대기 사용

promise보다는 비동기/대기 사용을 고려해야 합니다. 비동기 함수는 promise를 사용하는 것보다 간단하고 보일러플레이트가 더 적게 필요합니다. 대기는 비동기적으로 값을 기다리기 위해 비동기 함수에서만 사용할 수 있습니다.

다음 예에서는 async/await를 사용하여 `us-west-2`의 모든 Amazon DynamoDB 테이블을 나열합니다.

> **Note**
>
> 이 예를 실행하려면 다음을 수행합니다.
>
> - 프로젝트의 명령줄에 를 `npm install @aws-sdk/client-dynamodb` 입력하여 AWS SDK for JavaScript DynamoDB 클라이언트를 설치합니다.
> - AWS 자격 증명을 올바르게 구성했는지 확인하십시오. 자세한 설명은 자격 증명 설정 섹션을 참조하세요.

```
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**
>
> 모든 브라우저가 async/await를 지원하는 것은 아닙니다. async/await를 지원하는 브라우저 목록은 Async 함수를 참조하세요.

JavaScript 프라미스 사용하기

콜백을 사용하는 대신 서비스 클라이언트의 AWS SDK for JavaScript v3 메서드(`ListTablesCommand`) 를 사용하여 서비스를 호출하고 비동기 흐름을 관리할 수 있습니다. 다음 예는 `us-west-2`에서 Amazon DynamoDB 테이블의 이름을 가져오는 방법을 보여줍니다.

```
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);
  });
```

여러 프라미스를 조정하세요.

경우에 따라 코드는 여러 비동기식 호출이 모두 성공적으로 반환된 경우에만 조치가 필요한 여러 비동기식 호출을 수행해야 합니다. promise를 사용하지 않고 개별 비동기 메서드 호출을 관리하는 경우 `all` 메서드를 사용하는 추가 promise를 생성할 수 있습니다.

이 메서드는 메서드에 전달한 promise 배열이 이행되는 경우 umbrella promise를 이행합니다. 콜백 함수는 `all` 메서드에 전달되는 promises의 값 배열로 전달됩니다.

다음 예제에서 AWS Lambda 함수는 Amazon DynamoDB에 대해 세 번의 비동기 호출을 수행해야 하지만 각 호출에 대한 약속이 충족된 후에만 완료할 수 있습니다.

```
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;
```

promise에 대한 브라우저 및 Node.js 지원

네이티브 JavaScript 프라미스 (ECMAScript 2015) 에 대한 지원은 코드가 JavaScript 실행되는 엔진 및 버전에 따라 달라집니다. 코드를 실행해야 하는 각 환경에서 JavaScript 프라미스가 지원되는지 확인하려면 의 ECMAScript 호환성 표를 참조하십시오. GitHub

익명 콜백 함수를 사용하세요.

각 서비스 객체 메서드는 익명 콜백 함수를 마지막 파라미터로 수락할 수 있습니다. 이 콜백 함수의 시그니처는 다음과 같습니다.

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

이 콜백 함수는 성공적인 응답 또는 오류 데이터 반환 시 실행됩니다. 메서드 호출에 성공하면 `data` 파라미터에서 응답 내용을 콜백 함수에 사용할 수 있습니다. 호출에 실패하면 `error` 파라미터에 자세한 실패 정보가 제공됩니다.

일반적으로 콜백 함수 내 코드는 오류가 있는지 테스트하는데, 오류가 반환되면 처리합니다. 오류가 반환되지 않으면 코드는 응답의 data 파라미터에서 데이터를 검색합니다. 콜백 함수의 기본 형식은 다음 예제와 같습니다.

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

이전 예제에서는 오류 또는 반환되는 데이터에 대한 자세한 내용이 콘솔에 로깅됩니다. 다음은 서비스 객체에 대한 메서드 호출의 일부로 전달되는 콜백 함수를 보여주는 예제입니다.

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

서비스 클라이언트 요청 생성

AWS 서비스 클라이언트에 요청하는 것은 간단합니다. SDK 버전 3 (V3) 을 JavaScript 사용하면 요청을 보낼 수 있습니다.

> **Note**
>
> SDK 버전 3을 사용할 때 버전 2 (V2) 명령을 사용하여 작업을 수행할 수도 있습니다. JavaScript 자세한 설명은 V2 명령 사용 섹션을 참조하세요.

요청을 보내려면 다음을 수행합니다.

1. 특정 AWS 리전과 같이 원하는 구성으로 클라이언트 객체를 초기화합니다.

2. (선택 사항) 특정 Amazon S3 버킷 이름과 같은 요청 값을 사용하여 요청 JSON 객체를 생성합니다. 클라이언트 메서드와 연결된 이름을 가진 인터페이스의 API 참조 주제를 살펴봄으로써 요청의 파라미터를 검토할 수 있습니다. 예를 들어 *AbcCommand*클라이언트 메서드를 사용하는 경우 요청 인터페이스는 입니다. *AbcInput*
3. 필요에 따라 요청 객체를 입력으로 사용하여 서비스 명령을 초기화합니다.
4. 명령 객체를 입력으로 사용하여 클라이언트에서 send를 직접적으로 호출합니다.

예를 들어 us-west-2의 Amazon DynamoDB 테이블을 나열하려면 async/await를 사용하면 됩니다.

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

서비스 클라이언트 응답 처리

서비스 클라이언트 메서드는 직접적으로 호출되면 클라이언트 메서드와 연결된 이름을 가진 인터페이스의 응답 객체 인스턴스를 반환합니다. 예를 들어 *AbcCommand*클라이언트 메서드를 사용하는 경우 응답 객체는 *AbcResponse*(인터페이스) 유형입니다.

응답에서 반환된 액세스 데이터

응답 객체에는 서비스 요청에서 반환한 데이터가 속성으로 포함됩니다.

서비스 클라이언트 요청 생성에서 ListTablesCommand 명령은 응답의 TableNames 속성에 테이블 이름을 반환했습니다.

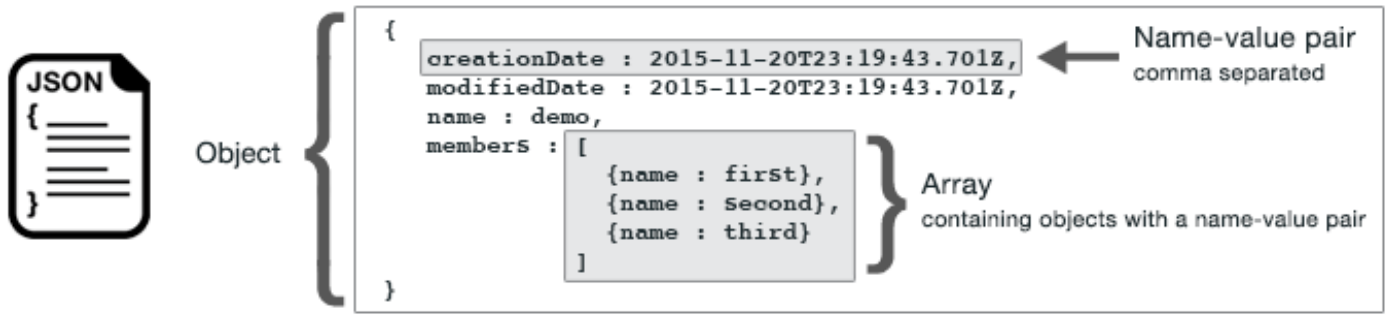
액세스 오류 정보

명령이 실패하면 예외가 발생합니다. 필요에 따라 예외를 처리할 수 있습니다.

JSON으로 작업하기

JSON은 인간과 머신이 둘 다 판독 가능한 데이터를 교환하기 위한 형식입니다. JSON이라는 이름은 JavaScript 객체 표기법의 약자이지만 JSON의 형식은 모든 프로그래밍 언어와 무관합니다.

는 요청을 할 때 JSON을 AWS SDK for JavaScript 사용하여 서비스 개체에 데이터를 보내고 서비스 개체의 데이터를 JSON으로 받습니다. JSON에 대한 자세한 내용은 json.org를 참조하세요.



JSON은 다음 두 가지 방식으로 데이터를 나타냅니다.

- 객체: 순서가 지정되지 안은 이름-값 쌍 모음. 객체는 여는 중괄호({)와 닫는 중괄호(}) 내에서 정의됩니다. 각 이름-값 쌍은 이름으로 시작하고 뒤에 콜론과 값이 옵니다. 이름-값 페어는 쉼표로 구분됩니다.
- 배열: 순서가 지정된 값 모음. 배열은 여는 대괄호([)와 닫는 대괄호(]) 안에 정의됩니다. 배열의 항목들은 쉼표로 구분됩니다.

다음은 객체가 카드 게임의 카드로 표현되는 객체 배열이 포함된 JSON 객체의 예입니다. 각 카드는 두 개의 이름-값 페어로 정의되는데, 하나는 하드를 식별하기 위한 고유한 값을 지정하고, 다른 하나는 해당하는 카드 이미지를 가리키는 URL을 지정합니다.

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

```
];
```

서비스 객체 파라미터로서 JSON

다음은 AWS Lambda 서비스 객체에 대한 직접 호출의 파라미터를 정의하는 데 사용되는 간단한 JSON의 예입니다.

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

params 객체는 여는 중괄호와 닫는 중괄호 내에서 쉼표로 구분된 이름-값 페어 3개로 정의됩니다. 서비스 객체 메서드에 파라미터를 제공하는 경우 이름은 호출하려는 서비스 객체 메서드에 대한 파라미터 이름으로 결정됩니다. Lambda 함수를 간접적으로 호출할 때 FunctionName, Payload, LogType은 Lambda 서비스 객체에서 invoke 메서드를 직접적으로 호출하는 데 사용되는 파라미터입니다.

서비스 객체 메서드 호출에 파라미터를 전달할 때 Lambda 함수를 호출하는 다음 예와 같이 메서드 호출에 JSON 객체를 제공합니다.

```
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 };
};
```

JavaScript 코드 예제를 위한 SDK

이 섹션의 항목에는 다양한 서비스의 AWS SDK for JavaScript API와 함께 를 사용하여 일반적인 작업을 수행하는 방법에 대한 예제가 포함되어 있습니다.

코드 예제 리포지토리에서 이러한 예제 및 기타 예제의 소스AWS 코드를 찾을 수 GitHub 있습니다. AWS 문서 팀이 제작을 고려할 새 코드 예제를 제안하려면 요청을 작성하세요. 이 팀은 개별 API 호출만 다루는 간단한 코드 조각에 비해 광범위한 시나리오 및 사용 사례를 다루는 코드를 생성하려고 합니다. 자세한 지침은 에 있는 기여 지침의 작성 코드 섹션을 참조하십시오. GitHub

Important

이러한 예에서는 ECMAScript6 import/export 구문을 사용합니다.

- 따라서 Node.js 버전 14.17 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
- CommonJS 구문을 사용하려는 경우 변환 지침은 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

주제

- JavaScript ES6/CommonJS 구문
- Amazon DynamoDB 예
- AWS Elemental MediaConvert 예제
- AWS Lambda 예제
- Amazon Lex 예
- Amazon Polly 예
- Amazon Redshift 예
- Amazon Simple Email Service 예
- Amazon Simple Notification Service 예
- Amazon Transcribe 예
- Amazon EC2 인스턴스에서 Node.js 설정
- DynamoDB에 데이터를 제출하는 앱 빌드
- 인증된 사용자로 트랜스크립션 앱 빌드
- API Gateway를 사용하여 Lambda 호출
- AWS SDK for JavaScript를 사용하여 AWS 서버리스 워크플로 생성
- AWS Lambda 함수를 실행하도록 예약된 이벤트 생성

- Amazon Lex 챗봇 빌드
- 메시징 애플리케이션 예 생성

JavaScript ES6/CommonJS 구문

AWS SDK for JavaScript 코드 예는 ECMAScript 6(ES6)로 작성되었습니다. ES6는 코드를 더 현대적이고 읽기 쉽게 만들고 더 많은 작업을 수행할 수 있도록 새로운 구문과 새로운 기능을 제공합니다.

ES6에서는 Node.js 버전 13.x 이상을 사용해야 합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 원하는 경우 다음 지침을 사용하여 코드 예를 CommonJS 구문으로 변환할 수 있습니다.

- 프로젝트 환경의 `package.json`에서 `"type" : "module"`을 제거합니다.
- 모든 ES6 `import` 문을 CommonJS `require` 문으로 변환합니다. 예를 들어 다음과 같이 변환합니다.

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

  위 구문을 동등한 다음 CommonJS 문으로 변환합니다.

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

- 모든 ES6 `export` 문을 CommonJS `module.exports` 문으로 변환합니다. 예를 들어 다음과 같이 변환합니다.

  ```
  export {s3}
  ```

  위 구문을 동등한 다음 CommonJS 문으로 변환합니다.

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

다음 예에서는 ES6와 CommonJS 모두에서 Amazon S3 버킷을 생성하는 코드 예를 보여줍니다.

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();
```

Amazon DynamoDB 예

Amazon DynamoDB는 문서 모델과 키-값 스토어 모델을 모두 지원하는 완전 관리형 NoSQL 클라우드 데이터베이스입니다. 전용 데이터베이스 서버를 프로비저닝하거나 유지 관리할 필요 없이 데이터에 대해 스키마 없는 테이블을 생성합니다.

DynamoDB용 JavaScript API는 `DynamoDB`, `DynamoDBStreams` 및 `DynamoDB.DocumentClient` 클라이언트 클래스를 통해 노출됩니다. DynamoDB 클라이언트 클래스 사용에 관한 자세한 내용은 API 참조의 Class: DynamoDB, Class: DynamoDBStreams 및 Class: DynamoDB utility를 참조하세요.

주제

- DynamoDB에서 테이블 생성 및 사용
- DynamoDB에서 단일 항목 읽기 및 쓰기
- DynamoDB에서 배치로 항목 읽기 및 쓰기
- DynamoDB 테이블 쿼리 및 스캔
- DynamoDB 문서 클라이언트 사용

DynamoDB에서 테이블 생성 및 사용

이 Node.js 코드 예제는 다음을 보여 줍니다.

- DynamoDB에서 데이터를 저장하고 검색하는 데 사용되는 테이블을 생성하고 관리하는 방법

시나리오

다른 데이터베이스 시스템과 마찬가지로 DynamoDB는 데이터를 테이블에 저장합니다. DynamoDB 테이블은 행과 유사한 항목으로 구성되는 데이터의 모음입니다. DynamoDB에서 데이터를 저장하거나 데이터에 액세스하려면 테이블을 생성하고 테이블로 작업합니다.

이 예에서는 일련의 Node.js 모듈을 사용하여 DynamoDB 테이블의 기본 작업을 수행합니다. 이 코드는 SDK for JavaScript에서 DynamoDB 클라이언트 클래스의 다음 메서드를 사용하여 테이블을 생성하고 테이블로 작업합니다.

- `CreateTableCommand`
- `ListTablesCommand`
- `DescribeTableCommand`
- `DeleteTableCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 다음 작업을 완료합니다.

- 이러한 Node.js 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- SDK for JavaScript DynamoDB 클라이언트를 설치합니다. 자세한 내용은 버전 3의 새 기능 섹션을 참조하세요.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

> **Note**
>
> 이 예에 사용된 데이터 형식에 관한 자세한 내용은 Amazon DynamoDB에서 지원되는 데이터 형식 및 명명 규칙을 참조하세요.

테이블 생성

파일 이름이 `create-table.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. 테이블을 생성하는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예제에서는 각 속성의 이름과 데이터 형식, 키 스키마, 테이블의 이름, 프로비저닝할 처리량의 단위가 포함됩니다. DynamoDB 서비스 객체의 `CreateTableCommand` 메서드를 직접적으로 호출합니다.

```
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;
```

```
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

테이블 목록 조회

파일 이름이 `list-tables.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. 테이블을 나열하는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다, 이 예제에서는 나열되는 테이블 수를 10으로 제한합니다. DynamoDB 서비스 객체의 `ListTablesCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

테이블 설명

파일 이름이 `describe-table.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. DynamoDB 서비스 객체의 `DescribeTableCommand` 메서드를 설명하는 데 필요한 파라미터가 포함된 JSON 객체를 생성합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

테이블 삭제

파일 이름이 `delete-table.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. 테이블을 삭제하는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예에서는 명령줄 파라미터로 제공되는 테이블의 이름을 포함합니다. DynamoDB 서비스 객체의 `DeleteTableCommand` 메서드를 직접적으로 호출합니다.

```
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;
```

```
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

DynamoDB에서 단일 항목 읽기 및 쓰기

이 Node.js 코드 예제는 다음을 보여 줍니다.

- DynamoDB 테이블에서 항목을 추가하는 방법
- DynamoDB 테이블에서 항목을 검색하는 방법
- DynamoDB 테이블에서 항목을 삭제하는 방법

시나리오

이 예에서는 일련의 Node.js 모듈에서 DynamoDB 클라이언트 클래스의 다음 메서드를 사용하여 DynamoDB 테이블에서 항목 하나를 읽고 씁니다.

- PutItemCommand
- UpdateItemCommand
- GetItemCommand
- DeleteItemCommand

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 다음 작업을 완료합니다.

- 이러한 Node.js 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.

- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.
- 액세스할 수 있는 항목이 포함된 DynamoDB 테이블을 생성합니다. DynamoDB 테이블 생성에 관한 자세한 내용은 DynamoDB에서 테이블 생성 및 사용 단원을 참조하세요.

> **Important**
>
> 이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

> **Note**
>
> 이 예에 사용된 데이터 형식에 관한 자세한 내용은 Amazon DynamoDB에서 지원되는 데이터 형식 및 명명 규칙을 참조하세요.

항목 쓰기

파일 이름이 `put-item.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. 항목을 추가하는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예제에서는 테이블의 이름 및 설정할 속성과 각 속성의 값을 정의하는 맵이 포함됩니다. DynamoDB 클라이언트 서비스 객체의 `PutItemCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

항목 업데이트

파일 이름이 `update-item.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 `DynamoDB` 클라이언트 서비스 객체를 생성합니다. 항목을 추가하는 데 필요한 파라미터가 포함된 JSON 객체를 생성합니다. 이 예에서는 객체에 테이블 이름, 업데이트할 키, 새 속성 이름을 매핑하는 날짜 표현식, 각각의 새 속성 값이 포함됩니다. DynamoDB 클라이언트 서비스 객체의 `UpdateItemCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

항목 가져오기

파일 이름이 `get-item.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. 가져올 항목을 식별하려면 테이블의 해당 항목에 대한 기본 키의 값을 제공해야 합니다. 기본적으로 `GetItemCommand` 메서드는 항목에 정의된 모든 속성 값을 반환합니다. 가능한 모든 속성 값의 일부만 가져오려면 프로젝션 표현식을 지정합니다.

항목을 가져오는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예제에서는 테이블의 이름, 가져오는 항목에 대한 키의 이름과 값, 검색할 항목 속성을 식별하는 프로젝션 표현식이 포함됩니다. DynamoDB 클라이언트 서비스 객체의 `GetItemCommand` 메서드를 직접적으로 호출합니다.

다음 코드 예에서는 파티션 키와 정렬 키 모두가 아니라 파티션 키로만 구성된 프라이머리 키가 포함된 테이블에서 항목을 검색합니다. 테이블에 파티션 키와 정렬 키로 구성된 프라이머리 키가 있는 경우 정렬 키 이름 및 속성도 지정해야 합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

항목 삭제

파일 이름이 `delete-item.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. 항목을 삭제하는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예에서는 테이블의 이름과 삭제하려는 항목의 키 이름 및 값을 포함합니다. DynamoDB 클라이언트 서비스 객체의 `DeleteItemCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node delete-item.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

DynamoDB에서 배치로 항목 읽기 및 쓰기

이 Node.js 코드 예제는 다음을 보여 줍니다.

- DynamoDB 테이블에서 항목의 배치를 읽고 쓰는 방법

시나리오

이 예에서는 일련의 Node.js 모듈을 사용하여 항목의 배치를 DynamoDB 테이블에 넣고 항목의 배치를 읽습니다. 이 코드는 SDK for JavaScript에서 DynamoDB 클라이언트 클래스의 다음 메서드를 사용하여 배치 읽기 및 쓰기 작업을 수행합니다.

- BatchGetItemCommand
- BatchWriteItemCommand

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 다음 작업을 완료합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.
- 액세스할 수 있는 항목이 포함된 DynamoDB 테이블을 생성합니다. DynamoDB 테이블 생성에 관한 자세한 내용은 DynamoDB에서 테이블 생성 및 사용 단원을 참조하세요.

Important

이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

Note

이 예에 사용된 데이터 형식에 관한 자세한 내용은 Amazon DynamoDB에서 지원되는 데이터 형식 및 명명 규칙을 참조하세요.

배치로 항목 읽기

파일 이름이 `batch-get-item.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 `DynamoDB` 클라이언트 서비스 객체를 생성합니다. 항목의 배치를 가져오는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예제에서는 읽을 하나 이상 테이블의 이름, 각 테이블에서 읽을 키의 값, 반환할 속성을 지정하는 프로젝션 표현식이 포함됩니다. DynamoDB 서비스 객체의 `BatchGetItemCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

배치로 항목 쓰기

파일 이름이 `batch-write-item.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. 항목의 배치를 가져오는 데 필요한 파라미터가 포함된 JSON 객체를 생성합니다. 이 예에서는 객체에 항목을 쓰려는 테이블, 각 항목에 대해 쓰려는 키,

속성 및 해당 값이 포함됩니다. DynamoDB 서비스 객체의 BatchWriteItemCommand 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node batch-write-item.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

DynamoDB 테이블 쿼리 및 스캔

이 Node.js 코드 예제는 다음을 보여 줍니다.

- DynamoDB 테이블에서 항목을 쿼리하고 스캔하는 방법

시나리오

쿼리는 기본 키 속성 값만 사용하여 테이블 또는 보조 인덱스에서 항목을 찾습니다. 검색할 파티션 키 이름과 값을 제공해야 합니다. 정렬 키 이름과 값도 제공할 수 있으며, 비교 연산자를 사용하여 검색 결과를 좁힐 수 있습니다. 스캔은 지정된 테이블에서 모든 항목을 확인하여 항목을 찾습니다.

이 예에서는 일련의 Node.js 모듈을 사용하여 DynamoDB 테이블에서 검색하려는 항목을 하나 이상 식별합니다. 이 코드는 SDK for JavaScript에서 DynamoDB 클라이언트 클래스의 다음 메서드를 사용하여 테이블을 쿼리하고 스캔합니다.

- QueryCommand
- ScanCommand

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 다음 작업을 완료합니다.

- 이러한 Node.js 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.
- 액세스할 수 있는 항목이 포함된 DynamoDB 테이블을 생성합니다. DynamoDB 테이블 생성에 관한 자세한 내용은 DynamoDB에서 테이블 생성 및 사용 단원을 참조하세요.

Important

이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

Note

이 예에 사용된 데이터 형식에 관한 자세한 내용은 Amazon DynamoDB에서 지원되는 데이터 형식 및 명명 규칙을 참조하세요.

테이블 쿼리

파일 이름이 `query.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 `DynamoDB` 클라이언트 서비스 객체를 생성합니다. 테이블을 쿼리하는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예제에서는 테이블 이름, 쿼리에 필요한 `ExpressionAttributeValues`, 해당 값을 사용하여 쿼리에서 반환되는 항목을 정의하는 `KeyConditionExpression`, 각 항목에 대해 반환할 속성 값의 이름이 포함됩니다. DynamoDB 서비스 객체의 `QueryCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node query.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

테이블 스캔

파일 이름이 `scan.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. DynamoDB에 액세스하려면 DynamoDB 클라이언트 서비스 객체를 생성합니다. 테이블에서 항목을 스캔하는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예제에서는 테이블의 이름, 일치하는 각 항목에 대해 반환할 속성 값의 목록, 결과 집합을 필터링하여 지정된 어구가 포함된 항목을 찾는 표현식이 포함됩니다. DynamoDB 서비스 객체의 `ScanCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node scan.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

DynamoDB 문서 클라이언트 사용

이 Node.js 코드 예제는 다음을 보여 줍니다.

- DynamoDB 유틸리티를 사용하여 DynamoDB 테이블에 액세스하는 방법

시나리오

DynamoDB 문서 클라이언트는 속성 값의 개념을 추상화하여 항목 작업을 간소화합니다. 이 추상화는 입력 파라미터로 제공되는 기본 JavaScript 유형에 주석을 달고, 주석이 달린 응답 데이터를 기본 JavaScript 유형으로 변환합니다.

DynamoDB 문서 클라이언트에 관한 자세한 내용은 GitHub의 @aws-sdk/lib-dynamodb README를 참조하세요. Amazon DynamoDB를 사용한 프로그래밍에 관한 자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB를 사용한 프로그래밍을 참조하세요.

이 예에서는 일련의 Node.js 모듈에서 DynamoDB 유틸리티를 사용하여 DynamoDB 테이블의 기본 작업을 수행합니다. 이 코드는 SDK for JavaScript에서 DynamoDB 문서 클라이언트 클래스의 다음 메서드를 사용하여 테이블을 쿼리하고 스캔합니다.

- GetCommand
- PutCommand
- UpdateCommand
- QueryCommand
- DeleteCommand

DynamoDB 문서 클라이언트 구성에 대한 자세한 내용은 @aws-sdk/lib-dynamodb를 참조하세요.

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 다음 작업을 완료합니다.

- 이러한 Node.js 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.
- 액세스할 수 있는 항목이 포함된 DynamoDB 테이블을 생성합니다. SDK for JavaScript를 사용하여 DynamoDB 테이블을 생성하는 방법에 관한 자세한 내용은 DynamoDB에서 테이블 생성 및 사용 단원을 참조하세요. DynamoDB 콘솔을 사용하여 테이블을 생성할 수도 있습니다.

> **Important**
>
> 이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

> **Note**
>
> 이 예에 사용된 데이터 형식에 관한 자세한 내용은 Amazon DynamoDB에서 지원되는 데이터 형식 및 명명 규칙을 참조하세요.

테이블에서 항목 가져오기

파일 이름이 `get.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 여기에는 `@aws-sdk/client-dynamodb`에 문서 클라이언트 기능을 제공하는 라이브러리 패키지인 `@aws-sdk/lib-dynamodb`가 포함됩니다. 그런 다음, 문서 클라이언트를 생성하는 동안 마샬링 및 마샬링 취소를 위해 아래와 같이 선택적인 두 번째 파라미터로 구성을 설정합니다. 다음으로, 클라이언트를 생성합니다. 이제 테이블에서 항목을 가져오는 데 필요한 파라미터가 포함된 JSON 객체를 생성합니다. 이 예에서는 객체에 테이블 이름, 해당 테이블의 해시 키 이름, 가져오려는 항목에 대한 해시 키의 값이 포함됩니다. DynamoDB 문서 클라이언트의 `GetCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node get.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

테이블에 항목 넣기

파일 이름이 put.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 여기에는 @aws-sdk/client-dynamodb에 문서 클라이언트 기능을 제공하는 라이브러리 패키지인 @aws-sdk/lib-dynamodb가 포함됩니다. 그런 다음, 문서 클라이언트를 생성하는 동안 마샬링 및 마샬링 취소를 위해 아래와 같이 선택적인 두 번째 파라미터로 구성을 설정합니다. 다음으로, 클라이언트를 생성합니다. 항목을 테이블에 쓰는 데 필요한 파라미터가 포함된 JSON 객체를 생성합니다. 이 예에서는 객체에 테이블 이름, 해시 키 및 값을 포함하여 추가하거나 업데이트할 항목에 대한 설명, 항목에 설정할 속성의 이름 및 값이 포함됩니다. DynamoDB 문서 클라이언트의 PutCommand 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node put.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

테이블에서 항목 업데이트

파일 이름이 update.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 여기에는 @aws-sdk/client-dynamodb에 문서

클라이언트 기능을 제공하는 라이브러리 패키지인 @aws-sdk/lib-dynamodb가 포함됩니다. 그런 다음, 문서 클라이언트를 생성하는 동안 마샬링 및 마샬링 취소를 위해 아래와 같이 선택적인 두 번째 파라미터로 구성을 설정합니다. 다음으로, 클라이언트를 생성합니다. 항목을 테이블에 쓰는 데 필요한 파라미터가 포함된 JSON 객체를 생성합니다. 이 예에서는 객체에 테이블 이름, 업데이트할 항목의 키, ExpressionAttributeValues 파라미터에서 값을 할당하는 토큰을 사용하여 업데이트할 항목의 속성을 정의하는 UpdateExpressions 집합이 포함됩니다. DynamoDB 문서 클라이언트의 UpdateCommand 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node update.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

테이블 쿼리

파일 이름이 query.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 여기에는 @aws-sdk/client-dynamodb에 문서 클라

이언트 기능을 제공하는 라이브러리 패키지인 @aws-sdk/lib-dynamodb가 포함됩니다. 테이블을 쿼리하는 데 필요한 파라미터를 포함하는 JSON 객체를 생성합니다. 이 예제에서는 테이블 이름, 쿼리에 필요한 ExpressionAttributeValues, 해당 값을 사용하여 쿼리에서 반환되는 항목을 정의하는 KeyConditionExpression이 포함됩니다. DynamoDB 문서 클라이언트의 QueryCommand 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node query.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

테이블에서 항목 삭제

파일 이름이 delete.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 여기에는 @aws-sdk/client-dynamodb에 문서 클라이언트 기능을 제공하는 라이브러리 패키지인 @aws-sdk/lib-dynamodb가 포함됩니다. 그런 다음, 문서 클라이언트를 생성하는 동안 마샬링 및 마샬링 취소를 위해 아래와 같이 선택적인 두 번

째 파라미터로 구성을 설정합니다. 다음으로, 클라이언트를 생성합니다. DynamoDB에 액세스하려면 DynamoDB 객체를 생성합니다. 테이블에서 항목을 삭제하는 데 필요한 파라미터가 포함된 JSON 객체를 생성합니다. 이 예에서는 객체에 테이블 이름, 삭제하려는 항목의 해시 키 이름 및 값이 포함됩니다. DynamoDB 문서 클라이언트의 `DeleteCommand` 메서드를 직접적으로 호출합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node delete.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

AWS Elemental MediaConvert 예제

AWS Elemental MediaConvert는 브로드캐스트급 기능을 갖춘 파일 기반의 비디오 트랜스코딩 서비스입니다. 이 서비스를 사용하여 인터넷에서 브로드캐스트 및 비디오 온디맨드(VOD) 전달용 자산을 생성할 수 있습니다. 자세한 정보는 AWS Elemental MediaConvert 사용 설명서를 참조하세요.

MediaConvert용 JavaScript API는 `MediaConvert` 클라이언트 클래스를 통해 노출됩니다. 자세한 내용은 API 참조의 Class: MediaConvert를 참조하세요.

주제

- MediaConvert의 리전별 엔드포인트 가져오기
- MediaConvert에서 트랜스코딩 작업 생성 및 관리
- MediaConvert에서 작업 템플릿 사용

MediaConvert의 리전별 엔드포인트 가져오기

이 Node.js 코드 예제는 다음을 보여 줍니다.

- MediaConvert에서 리전별 엔드포인트를 검색하는 방법

시나리오

이 예에서는 Node.js 모듈을 사용하여 MediaConvert를 직접적으로 호출하고 리전별 엔드포인트를 검색합니다. 서비스 기본 엔드포인트에서 엔드포인트 URL을 검색할 수 있으며 이 작업을 수행하는 데는 리전별 엔드포인트가 필요하지 않습니다. 이 코드는 SDK for JavaScript에서 MediaConvert 클라이언트 클래스의 다음 메서드를 사용하여 이 엔드포인트를 검색합니다.

- `DescribeEndpointsCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 다음 작업을 완료합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.
- 출력 파일이 저장된 Amazon S3 버킷 및 입력 파일에 대한 액세스 권한을 MediaConvert에 부여하는 IAM 역할을 생성합니다. 자세한 내용은 AWS Elemental MediaConvert 사용 설명서의 Set up IAM permissions 단원을 참조하세요.

> **Important**
>
> 이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

엔드포인트 URL 가져오기

`libs` 디렉터리를 생성하고 파일 이름이 `emcClientGet.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 MediaConvert 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `emc_getendpoint.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다.

MediaConvert 클라이언트 클래스의 `DescribeEndpointsCommand` 메서드에 대한 비어 있는 요청 파라미터를 전달할 객체를 생성합니다. 그런 다음 `DescribeEndpointsCommand` 메서드를 호출합니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node emc_getendpoint.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

MediaConvert에서 트랜스코딩 작업 생성 및 관리

이 Node.js 코드 예제는 다음을 보여 줍니다.

- MediaConvert에 사용할 리전별 엔드포인트를 지정하는 방법
- MediaConvert에서 트랜스코딩 작업을 생성하는 방법
- 트랜스코딩 작업을 취소하는 방법.
- 완료된 트랜스코딩 작업에 대한 JSON을 검색하는 방법.
- 최근에 생성된 최대 20개 작업에 대한 JSON 배열을 검색하는 방법.

시나리오

이 예에서는 Node.js 모듈을 사용하여 트랜스코딩 작업을 생성하고 관리할 MediaConvert를 직접적으로 호출합니다. 이 코드는 SDK for JavaScript에서 MediaConvert 클라이언트 클래스의 다음 메서드를 사용하여 이 작업을 수행합니다.

- `CreateJobCommand`
- `CancelJobCommand`
- `GetJobCommand`
- `ListJobsCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 다음 작업을 완료합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.
- 작업 입력 파일 및 출력 파일을 위한 스토리지를 제공하는 Amazon S3 버킷을 생성하고 구성합니다. 자세한 내용은 AWS Elemental MediaConvert 사용 설명서의 Create storage for files 단원을 참조하세요.
- 입력 스토리지를 위해 프로비저닝한 Amazon S3 버킷에 입력 비디오를 업로드합니다. 지원되는 입력 비디오 코덱 및 컨테이너 목록은 AWS Elemental MediaConvert 사용 설명서의 Supported input codecs and containers 단원을 참조하세요.
- 출력 파일이 저장된 Amazon S3 버킷 및 입력 파일에 대한 액세스 권한을 MediaConvert에 부여하는 IAM 역할을 생성합니다. 자세한 내용은 AWS Elemental MediaConvert 사용 설명서의 Set up IAM permissions 단원을 참조하세요.

> **Important**
>
> 이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

SDK 구성

필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다. MediaConvert는 각 계정에 사용자 지정 엔드포인트를 사용하므로 `MediaConvert` 클라이언트 클래스도 리전별 엔드포인트를 사용하도록 구성해야 합니다. 이렇게 하려면 `mediaconvert(endpoint)`에서 `endpoint` 파라미터를 설정합니다.

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

단순 트랜스코딩 작업 정의

`libs` 디렉터리를 생성하고 파일 이름이 `emcClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 MediaConvert 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다. *`ENDPOINT`*를 MediaConvert 계정 엔드포인트로 바꿉니다. MediaConvert 콘솔의 계정 페이지에서 이 엔드포인트를 확인할 수 있습니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `emc_createjob.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 트랜스코드 작업 파라미터를 정의하는 JSON을 생성합니다.

이러한 파라미터는 매우 세부적입니다. AWS Elemental MediaConvert 콘솔을 사용하면 콘솔에서 작업 설정을 선택한 다음, 작업 섹션 하단에서 작업 JSON 표시를 선택하여 JSON 작업 파라미터를 생성할 수 있습니다. 이 예제는 단순 작업용 JSON을 보여줍니다.

> **Note**
>
> *`JOB_QUEUE_ARN`*을 MediaConvert 작업 대기열로 바꾸고, *`IAM_ROLE_ARN`*을 IAM 역할의 Amazon 리소스 이름(ARN)으로 바꾸며, *`OUTPUT_BUCKET_NAME`*을 대상 버킷 이름(예: "s3://OUTPUT_BUCKET_NAME/")으로 바꾸고, *`INPUT_BUCKET_AND_FILENAME`*을 입력 버킷 및 파일 이름(예: "s3://INPUT_BUCKET/FILE_NAME")으로 바꿉니다.

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

트랜스코딩 작업 생성

작업 파라미터 JSON을 생성한 후 비동기 `run` 메서드를 직접적으로 호출하여 `MediaConvert` 클라이언트 서비스 객체를 간접적으로 호출해서 파라미터를 전달합니다. 생성된 작업의 ID는 응답 `data`에서 반환됩니다.

```
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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node emc_createjob.js
```

이 전체 코드 예는 여기 GitHub에서 찾을 수 있습니다.

트랜스코딩 작업 취소

libs 디렉터리를 생성하고 파일 이름이 emcClient.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 MediaConvert 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다. *ENDPOINT*를 MediaConvert 계정 엔드포인트로 바꿉니다. MediaConvert 콘솔의 계정 페이지에서 이 엔드포인트를 확인할 수 있습니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 emc_canceljob.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 취소할 작업의 ID가 포함된 JSON을 생성합니다. 그런 다음, 파라미터를 전달하는 MediaConvert 클라이언트 서비스 객체를 간접적으로 호출하기 위한 promise를 생성하여 CancelJobCommand 메서드를 직접적으로 호출합니다. promise 콜백에서 응답을 처리합니다.

> **Note**
>
> *JOB_ID*를 취소할 작업의 ID로 바꿉니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node ec2_canceljob.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

최근 트랜스코딩 작업 나열

`libs` 디렉터리를 생성하고 파일 이름이 `emcClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 MediaConvert 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다. *ENDPOINT*를 MediaConvert 계정 엔드포인트로 바꿉니다. MediaConvert 콘솔의 계정 페이지에서 이 엔드포인트를 확인할 수 있습니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `emc_listjobs.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다.

목록을 `ASCENDING` 또는 `DESCENDING` 순서로 정렬할지 여부, 확인할 작업 대기열의 Amazon 리소스 이름(ARN), 포함할 작업의 상태 등을 지정하는 값을 포함하여 파라미터 JSON을 생성합니다. 그런 다음, 파라미터를 전달하는 `MediaConvert` 클라이언트 서비스 객체를 간접적으로 호출하기 위한 promise를 생성하여 `ListJobsCommand` 메서드를 직접적으로 호출합니다.

> **Note**
>
> *`QUEUE_ARN`*을 확인할 작업 대기열의 Amazon 리소스 이름(ARN)으로 바꾸고 *`STATUS`*를 대기열의 상태로 바꿉니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node emc_listjobs.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

MediaConvert에서 작업 템플릿 사용

이 Node.js 코드 예제는 다음을 보여 줍니다.

- AWS Elemental MediaConvert 작업 템플릿을 생성하는 방법.
- 작업 템플릿을 사용하여 트랜스코딩 작업을 생성하는 방법.
- 모든 작업 템플릿의 목록을 표시하는 방법.
- 작업 템플릿을 삭제하는 방법.

시나리오

MediaConvert에서 트랜스코딩 작업을 생성하는 데 필요한 JSON은 많은 수의 설정을 포함하여 세부적입니다. 후속 작업을 생성하는 데 사용할 수 있는 작업 템플릿에 알려진 좋은 설정을 저장하여 작업 생성을 대폭 간소화할 수 있습니다. 이 예에서는 Node.js 모듈을 사용해 MediaConvert를 직접적으로 호출하여 작업 템플릿을 생성, 사용 및 관리합니다. 이 코드는 SDK for JavaScript에서 MediaConvert 클라이언트 클래스의 다음 메서드를 사용하여 이 작업을 수행합니다.

- `CreateJobTemplateCommand`
- `CreateJobCommand`
- `DeleteJobTemplateCommand`
- `ListJobTemplatesCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 다음 작업을 완료합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.
- 출력 파일이 저장된 Amazon S3 버킷 및 입력 파일에 대한 액세스 권한을 MediaConvert에 부여하는 IAM 역할을 생성합니다. 자세한 내용은 AWS Elemental MediaConvert 사용 설명서의 Set up IAM permissions 단원을 참조하세요.

> **Important**
>
> 이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

작업 템플릿 생성

`libs` 디렉터리를 생성하고 파일 이름이 `emcClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 MediaConvert 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다. *`ENDPOINT`*를 MediaConvert 계정 엔드포인트로 바꿉니다. MediaConvert 콘솔의 계정 페이지에서 이 엔드포인트를 확인할 수 있습니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `emc_create_jobtemplate.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다.

템플릿 생성을 위한 파라미터 JSON을 지정합니다. 이전에 성공한 작업에서 대부분의 JSON 파라미터를 사용하여 템플릿에서 `Settings` 값을 지정할 수 있습니다. 이 예제에서는 MediaConvert에서 트랜스코딩 작업 생성 및 관리의 작업 설정을 사용합니다.

파라미터를 전달하는 `MediaConvert` 클라이언트 서비스 객체를 간접적으로 호출하기 위한 promise를 생성하여 `CreateJobTemplateCommand` 메서드를 직접적으로 호출합니다.

> **Note**
>
> *JOB_QUEUE_ARN*을 확인할 작업 대기열의 Amazon 리소스 이름(ARN)으로 바꾸고, *BUCKET_NAME*을 대상 Amazon S3 버킷의 이름(예: "s3://BUCKET_NAME/")으로 바꿉니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node emc_create_jobtemplate.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

작업 템플릿에서 트랜스코딩 작업 생성

`libs` 디렉터리를 생성하고 파일 이름이 `emcClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 MediaConvert 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다. *`ENDPOINT`*를 MediaConvert 계정 엔드포인트로 바꿉니다. MediaConvert 콘솔의 계정 페이지에서 이 엔드포인트를 확인할 수 있습니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 emc_template_createjob.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다.

사용할 작업 템플릿의 이름, 생성하는 작업에 특정하게 사용할 Settings를 포함하여 작업 생성 파라미터 JSON을 생성합니다. 그런 다음, 파라미터를 전달하는 MediaConvert 클라이언트 서비스 객체를 간접적으로 호출하기 위한 promise를 생성하여 CreateJobsCommand 메서드를 직접적으로 호출합니다.

> **Note**
>
> *JOB_QUEUE_ARN*을 확인할 작업 대기열의 Amazon 리소스 이름(ARN)으로 바꾸고, *KEY_PAIR_NAME*, *TEMPLATE_NAME*, *ROLE_ARN*을 역할의 Amazon 리소스 이름(ARN)으로 바꾸며, *INPUT_BUCKET_AND_FILENAME*을 입력 버킷 및 파일 이름(예: "s3://BUCKET_NAME/FILE_NAME")으로 바꿉니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node emc_template_createjob.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

작업 템플릿 목록 표시

`libs` 디렉터리를 생성하고 파일 이름이 `emcClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 MediaConvert 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다. *`ENDPOINT`*를 MediaConvert 계정 엔드포인트로 바꿉니다. MediaConvert 콘솔의 계정 페이지에서 이 엔드포인트를 확인할 수 있습니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 emc_listtemplates.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다.

MediaConvert 클라이언트 클래스의 listTemplates 메서드에 대한 요청 파라미터를 전달할 객체를 생성합니다. 나열할 템플릿(NAME, CREATION_DATE, SYSTEM), 나열할 개수 및 정렬 순서를 결정하는 값을 포함시킵니다. ListTemplatesCommand 메서드를 직접적으로 호출하려면 MediaConvert 서비스 객체를 간접적으로 호출하기 위한 promise를 생성하고 파라미터를 전달합니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node emc_listtemplates.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

작업 템플릿 삭제

libs 디렉터리를 생성하고 파일 이름이 emcClient.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 MediaConvert 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다. *ENDPOINT*를 MediaConvert 계정 엔드포인트로 바꿉니다. MediaConvert 콘솔의 계정 페이지에서 이 엔드포인트를 확인할 수 있습니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 emc_deletetemplate.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다.

삭제하려는 작업 템플릿의 이름을 MediaConvert 클라이언트 클래스의 DeleteJobTemplateCommand 메서드에 대한 파라미터로 전달할 객체를 생성합니다. DeleteJobTemplateCommand 메서드를 직접적으로 호출하려면 MediaConvert 서비스 객체를 간접적으로 호출하기 위한 promise를 생성하고 파라미터를 전달합니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node emc_deletetemplate.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

AWS Lambda 예제

AWS Lambda는 서버 프로비저닝 또는 관리, 워크로드 인식 클러스터 규모 조정 로직 생성, 이벤트 통합 유지 또는 런타임 관리 없이도 코드를 실행할 수 있게 지원하는 서버리스 컴퓨팅 서비스입니다.

AWS Lambda용 JavaScript API는 LambdaService 클라이언트 클래스를 통해 노출됩니다.

다음은 AWS SDK for JavaScript v3에서 Lambda 함수를 생성하고 사용하는 방법을 보여주는 예 목록입니다.

- API Gateway를 사용하여 Lambda 호출
- AWS Lambda 함수를 실행하도록 예약된 이벤트 생성

Amazon Lex 예

Amazon Lex는 음성 및 텍스트를 사용하는 애플리케이션에 대화형 인터페이스를 구축하기 위한 AWS 서비스입니다.

Amazon Lex용 JavaScript API는 Lex Runtime Service 클라이언트 클래스를 통해 노출됩니다.

- Amazon Lex 챗봇 빌드

Amazon Polly 예

이 Node.js 코드 예제는 다음을 보여 줍니다.

- Amazon S3에 Amazon Polly를 사용해 녹음한 오디오 업로드

시나리오

이 예에서는 일련의 Node.js 모듈에서 Amazon S3 클라이언트 클래스의 다음 메서드를 사용하여 Amazon Polly를 사용해 녹음한 오디오를 Amazon S3에 자동으로 업로드합니다.

- `StartSpeechSynthesisTaskCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- GitHub의 지침에 따라 노드 JavaScript 예를 실행하도록 프로젝트 환경을 설정합니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.
- AWS Identity and Access Management(IAM) 미인증 Amazon Cognito 사용자 역할 polly:SynthesizeSpeech 권한을 생성하고 이 권한에 IAM 역할이 연결된 Amazon Cognito 자격 증명 풀을 생성합니다. 아래 AWS CloudFormation을 사용하여 AWS 리소스 생성 단원에서는 이러한 리소스를 생성하는 방법을 설명합니다.

> **Note**
>
> 이 예에서는 Amazon Cognito를 사용하지만, Amazon Cognito를 사용하지 않는 경우 AWS 사용자에게 다음 IAM 권한 정책이 있어야 합니다.

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

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

AWS CloudFormation을 사용하여 AWS 리소스 생성

AWS CloudFormation을 사용하여 AWS 인프라 배포를 예상한 대로 반복해서 생성하고 프로비저닝할 수 있습니다. AWS CloudFormation에 대한 자세한 내용은 사용 설명서AWS CloudFormation를 참조하세요.

AWS CloudFormation 스택을 생성하려면 다음을 수행합니다.

1. AWS CLI 사용 설명서의 지침에 따라 AWS CLI를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 디렉터리에 이름이 `setup.yaml`인 파일을 생성하고 여기 GitHub의 내용을 해당 파일에 복사합니다.

> **Note**
>
> AWS CloudFormation 템플릿은 여기 GitHub에서 제공하는 AWS CDK를 사용하여 생성되었습니다. AWS CDK에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 안내서 단원을 참조하세요.

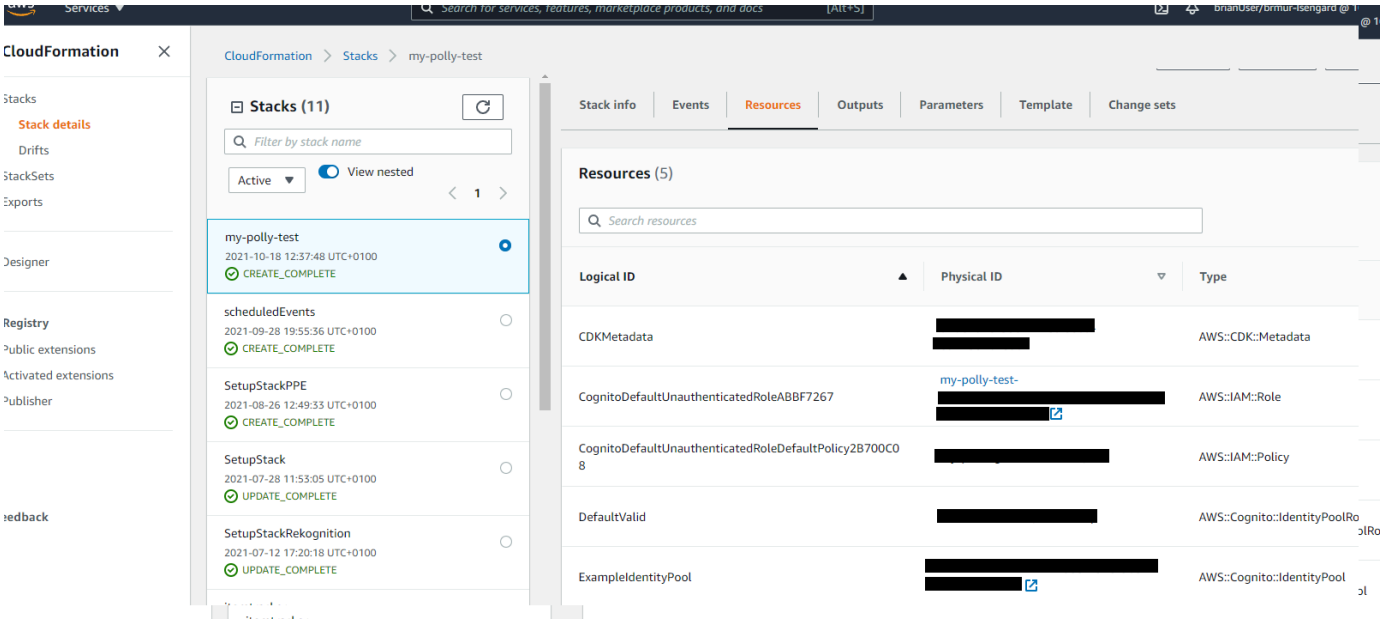
3. 명령줄에서 다음 명령을 실행하여 *`STACK_NAME`*을 스택의 고유한 이름으로 바꿉니다.

> **Important**
>
> 스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

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

`create-stack` 명령 파라미터에 대한 자세한 내용은 AWS CLI 명령 참조 가이드 및 AWS CloudFormation 사용 설명서를 참조하세요.

4. AWS CloudFormation 관리 콘솔로 이동하여 스택을 선택하고 스택 이름을 선택한 다음, 리소스 탭을 선택하여 생성된 리소스 목록을 확인합니다.



Amazon S3에 Amazon Polly를 사용해 녹음한 오디오 업로드

파일 이름이 `polly_synthesize_to_s3.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 코드에 *`REGION`* 및 *`BUCKET_NAME`*을 입력합니다. Amazon Polly에 액세스하려면 `Polly` 클라이언트 서비스 객체를 생성합니다. *`"IDENTITY_POOL_ID"`*를 이 예에서 생성한 Amazon Cognito 자격 증명 풀의 샘플 페이지에 있는 `IdentityPoolId`로 바꿉니다. 이는 각 클라이언트 객체에도 전달됩니다.

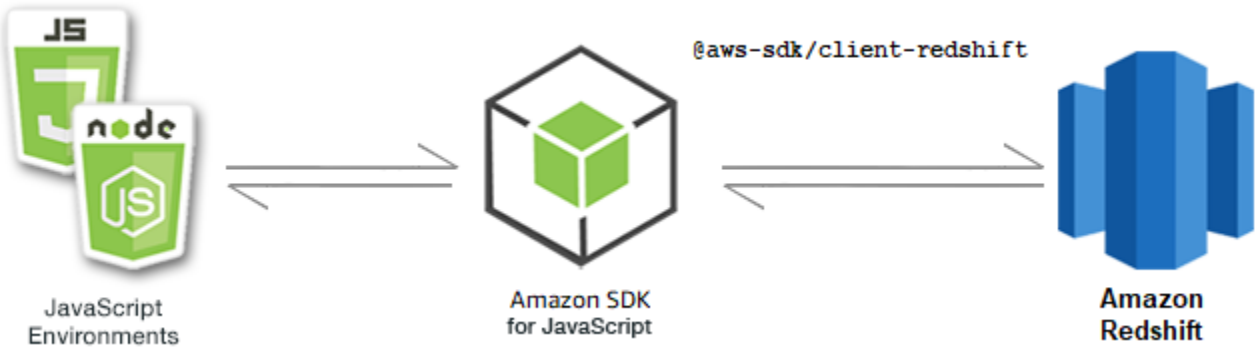
Amazon Polly 클라이언트 서비스 객체의 `StartSpeechSynthesisCommand` 메서드를 직접적으로 호출하여 음성 메시지를 합성해서 Amazon S3 버킷에 업로드합니다.

```
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();
```

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Redshift 예

Amazon Redshift는 클라우드에서 완벽하게 관리되는 페타바이트급 데이터 웨어하우스 서비스입니다. Amazon Redshift 데이터 웨어하우스는 노드라는 컴퓨팅 리소스의 모음으로, 노드는 클러스터라는 그룹을 구성합니다. 각 클러스터는 Amazon Redshift 엔진을 실행하며, 하나 이상의 데이터베이스를 포함합니다.



Amazon Redshift용 JavaScript API는 Amazon Redshift 클라이언트 클래스를 통해 노출됩니다.

주제

- Amazon Redshift 예

Amazon Redshift 예

이 예에서는 일련의 Node.js 모듈에서 `Redshift` 클라이언트 클래스의 다음 메서드를 사용하여 Amazon Redshift 클러스터의 파라미터를 생성, 수정, 설명하고 클러스터를 삭제합니다.

- `CreateClusterCommand`
- `ModifyClusterCommand`
- `DescribeClustersCommand`
- `DeleteClusterCommand`

Amazon Redshift 사용자에 관한 자세한 내용은 Amazon Redshift 시작 안내서를 참조하세요.

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.
>
> - 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> - CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

Amazon Redshift 클러스터 생성

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Redshift 클러스터를 생성하는 방법을 보여줍니다. 자세한 내용은 CreateCluster 단원을 참조하세요.

> **Important**
>
> 생성하려는 클러스터가 활성화됩니다(샌드박스에서 실행되지 않음). 클러스터를 삭제할 때까지 클러스터에 대해 기본 Amazon Redshift 사용 요금이 청구됩니다. 클러스터를 생성할 때와 같은 작업 기간 내에 해당 클러스터를 삭제하면 총 청구 비용이 가장 적게 듭니다.

`libs` 디렉터리를 생성하고 파일 이름이 `redshiftClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Redshift 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `redshift-create-cluster.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 프로비저닝할 노드 유형, 클러스터에 자동으로 생성되는 데이터베이스 인스턴스의 마스터 로그인 보안 인증, 마지막으로 클러스터 유형을 지정하여 파라미터 객체를 생성합니다.

> **Note**
>
> *CLUSTER_NAME*을 클러스터 이름으로 바꿉니다. *NODE_TYPE*의 경우 프로비저닝할 노드 유형(예: 'dc2.large')을 지정합니다. *MASTER_USERNAME* 및 *MASTER_USER_PASSWORD*는 클러스터에 있는 DB 인스턴스의 마스터 사용자 로그인 보안 인증 정보입니다. *CLUSTER_TYPE*에는 클러스터 유형을 입력합니다. `single-node`를 지정하는 경우 `NumberOfNodes` 파라미터가 필요하지 않습니다. 나머지 파라미터는 선택 사항입니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Redshift 클러스터 수정

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Redshift 클러스터의 마스터 사용자 암호를 수정하는 방법을 보여줍니다. 수정할 수 있는 다른 설정에 관한 자세한 내용은 ModifyCluster 단원을 참조하세요.

`libs` 디렉터리를 생성하고 파일 이름이 `redshiftClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Redshift 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `redshift-modify-cluster.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. AWS 리전, 수정하려는 클러스터 이름, 새 마스터 사용자 암호를 지정합니다.

> **Note**
>
> *`CLUSTER_NAME`*을 클러스터 이름으로 바꾸고 *`MASTER_USER_PASSWORD`*를 새 마스터 사용자 암호로 바꿉니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Redshift 클러스터의 세부 정보 보기

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Redshift 클러스터의 세부 정보를 확인하는 방법을 보여줍니다. 옵션에 관한 자세한 내용은 DescribeClusters 단원을 참조하세요.

`libs` 디렉터리를 생성하고 파일 이름이 `redshiftClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Redshift 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `redshift-describe-clusters.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. AWS 리전, 수정하려는 클러스터 이름, 새 마스터 사용자 암호를 지정합니다.

> **Note**
>
> *CLUSTER_NAME*을 클러스터 이름으로 바꿉니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Redshift 클러스터 삭제

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Redshift 클러스터의 세부 정보를 확인하는 방법을 보여줍니다. 수정할 수 있는 다른 설정에 관한 자세한 내용은 DeleteCluster 단원을 참조하세요.

`libs` 디렉터리를 생성하고 파일 이름이 `redshiftClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Redshift 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `redshift-delete-clusters.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. AWS 리전, 수정하려는 클러스터 이름, 새 마스터 사용자 암호를 지정합니다. 삭제하기 전에 클러스터의 최종 스냅샷을 저장할지 여부를 지정하고, 저장할 경우 스냅샷의 ID를 지정합니다.

> **Note**
>
> *CLUSTER_NAME*을 클러스터 이름으로 바꿉니다. *SkipFinalClusterSnapshot*의 경우 클러스터를 삭제하기 전에 해당 클러스터의 최종 스냅샷을 생성할지 여부를 지정합니다. 'false'를 지정하는 경우 *CLUSTER_SNAPSHOT_ID*에 최종 클러스터 스냅샷의 ID를 지정합니다. 클러스터 대시보드에서 클러스터에 대한 스냅샷 열의 링크를 클릭하고 스냅샷 창까지 아래로 스크롤하여 이 ID를 얻을 수 있습니다. `rs:` 스템은 스냅샷 ID의 일부가 아닙니다.

```
// 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();
```

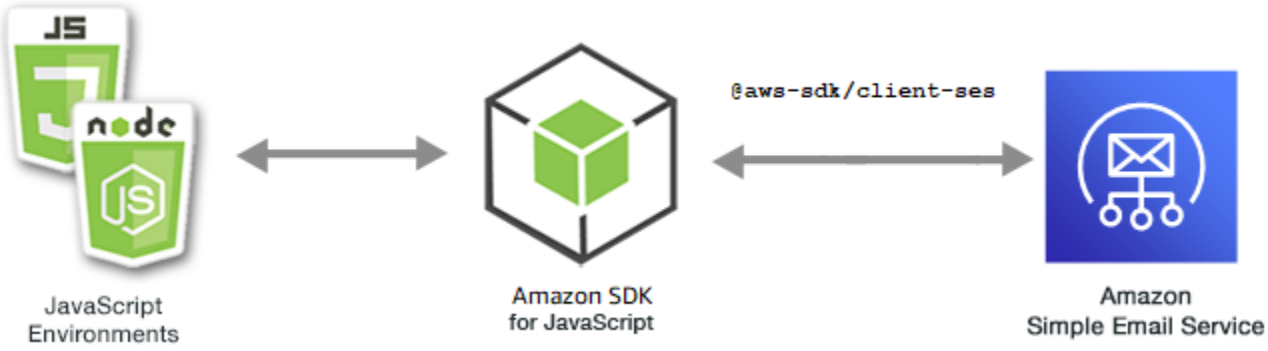
예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Simple Email Service 예

Amazon Simple Email Service(Amazon SES)는 디지털 마케팅 담당자 및 애플리케이션 개발자가 마케팅, 알림 및 거래 이메일을 전송하는 데 도움이 되도록 설계된 클라우드 기반 이메일 전송 서비스입니다. 이 서비스는 이메일을 사용하여 고객과 연락하는 모든 규모의 기업을 위한 안정적이고 비용 효과적인 서비스입니다.



Amazon JavaScript SES용 API는 SES 클라이언트 클래스를 통해 공개됩니다. Amazon SES 클라이언트 클래스 사용에 관한 자세한 내용은 API 참조의 Class: SES를 참조하세요.

주제

- Amazon SES 자격 증명 관리
- Amazon SES에서 이메일 템플릿 작업
- Amazon SES를 사용하여 이메일 전송

Amazon SES 자격 증명 관리

이 Node.js 코드 예제는 다음을 보여 줍니다.

- Amazon SES에 사용되는 이메일 주소 및 도메인을 확인하는 방법
- Amazon SES 자격 증명에 AWS Identity and Access Management (IAM) 정책을 할당하는 방법.
- AWS 계정의 모든 Amazon SES ID를 나열하는 방법.
- Amazon SES에 사용되는 자격 증명을 삭제하는 방법

Amazon SES 자격 증명은 Amazon SES에서 이메일을 보내는 데 사용하는 이메일 주소 또는 도메인입니다. Amazon SES에서는 이메일 자격 증명을 확인해야 합니다. 이렇게 해당 자격 증명을 소유하고 있음을 확인하고 다른 사람이 이를 사용하지 못하게 방지합니다.

Amazon SES에서 이메일 주소 및 도메인을 확인하는 방법에 대한 자세한 내용은 Amazon Simple Email Service 개발자 안내서의 Amazon SES에서 확인된 자격 증명 단원을 참조하세요. Amazon SES의 전송 권한 부여에 관한 자세한 내용은 Overview of Amazon SES sending authorization 단원을 참조하세요.

시나리오

이 예에서는 일련의 Node.js 모듈을 사용하여 Amazon SES 자격 증명을 확인하고 관리합니다. Node.js 모듈은 SDK JavaScript Form을 사용하여 SES 클라이언트 클래스의 다음 메서드를 사용하여 이메일 주소와 도메인을 확인합니다.

- `ListIdentitiesCommand`
- `DeleteIdentityCommand`
- `VerifyEmailIdentityCommand`
- `VerifyDomainIdentityCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 Node TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. 의 지침을 따르십시오 GitHub.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

Important

이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.

- 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
- CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

자격 증명 나열

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용할 이메일 주소와 도메인을 나열합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. *### ## ####* 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_listidentities.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

`SES` 클라이언트 클래스의 `ListIdentitiesCommand` 메서드에 대한 `IdentityType` 및 기타 파라미터를 전달할 객체를 생성합니다. `ListIdentitiesCommand` 메서드를 직접적으로 호출하려면 Amazon SES 서비스 객체를 간접적으로 호출하여 파라미터 객체를 전달합니다.

반환된 `data`에는 `IdentityType` 파라미터로 지정된 도메인 자격 증명 배열이 포함되어 있습니다.

Note

IDENTITY_TYPE# ID 유형 (예: EmailAddress "" 또는 “도메인”) 으로 바꾸십시오.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node ses_listidentities.js
```

이 예제 코드는 여기 에서 확인할 수 있습니다. GitHub

이메일 주소 자격 증명 확인

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용할 이메일 발신자를 확인합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. `### ## ####` 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_verifyemailidentity.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SES 클라이언트 클래스의 `VerifyEmailIdentityCommand` 메서드에 대한 `EmailAddress` 파라미터를 전달할 객체를 생성합니다. `VerifyEmailIdentityCommand` 메서드를 직접적으로 호출하려면 Amazon SES 클라이언트 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> *ADDRESS@DOMAIN.EXT*를 name@example.com과 같은 이메일 주소로 바꿉니다.

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

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. 도메인이 확인을 위해 Amazon SES에 추가됩니다.

```
node ses_verifyemailidentity.js
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

도메인 자격 증명 확인

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용할 이메일 도메인을 확인합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. *### ## ####* 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_verifydomainidentity.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SES 클라이언트 클래스의 `VerifyDomainIdentityCommand` 메서드에 대한 `Domain` 파라미터를 전달할 객체를 생성합니다. `VerifyDomainIdentityCommand` 메서드를 직접적으로 호출하려면 Amazon SES 클라이언트 서비스 객체를 간접적으로 호출하여 파라미터 객체를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 `send` 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *AMI_ID*를 실행할 Amazon Machine Image(AMI) ID로 바꾸고, AMI ID에 할당할 키 페어의 *KEY_PAIR_NAME*을 바꿉니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. 도메인이 확인을 위해 Amazon SES에 추가됩니다.

```
node ses_verifydomainidentity.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

자격 증명 삭제

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용되는 이메일 주소 또는 도메인을 삭제합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. ### ## #### 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_deleteidentity.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

`SES` 클라이언트 클래스의 `DeleteIdentityCommand` 메서드에 대한 `Identity` 파라미터를 전달할 객체를 생성합니다. `DeleteIdentityCommand` 메서드를 직접적으로 호출하려면 Amazon SES 클라이언트 서비스 객체를 간접적으로 호출하기 위한 `request`를 생성하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 `send` 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *`IDENTITY_TYPE`*을 삭제할 자격 증명 유형으로 바꾸고, *`IDENTITY_NAME`*을 삭제할 자격 증명 이름으로 바꿉니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node ses_deleteidentity.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

Amazon SES에서 이메일 템플릿 작업

이 Node.js 코드 예제는 다음을 보여 줍니다.

- 모든 이메일 템플릿 목록을 가져오는 방법
- 이메일 템플릿을 검색하고 업데이트하는 방법
- 이메일 템플릿을 생성하고 삭제하는 방법

Amazon SES에서 이메일 템플릿을 사용하여 맞춤형 이메일 메시지를 전송할 수 있습니다. Amazon SES에서 이메일 템플릿을 생성하고 사용하는 방법에 대한 자세한 내용은 Amazon Simple Email Service 개발자 안내서의 템플릿을 사용하여 Amazon SES API를 통해 맞춤형 이메일 전송 단원을 참조하세요.

시나리오

이 예제에서는 일련의 Node.js 모듈을 사용하여 이메일 템플릿을 작업합니다. Node.js 모듈은 SDK JavaScript Form을 사용하여 SES 클라이언트 클래스의 다음 메서드를 사용하여 이메일 템플릿을 만들고 사용합니다.

- `ListTemplatesCommand`

- `CreateTemplateCommand`
- `GetTemplateCommand`
- `DeleteTemplateCommand`
- `UpdateTemplateCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 Node TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. 의 지침을 따르십시오 GitHub.

- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.
>
> - 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> - CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

이메일 템플릿 나열

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용할 이메일 템플릿을 생성합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. `### ## ####` 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_listtemplates.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SES 클라이언트 클래스의 `ListTemplatesCommand` 메서드에 대한 파라미터를 전달할 객체를 생성합니다. `ListTemplatesCommand` 메서드를 직접적으로 호출하려면 Amazon SES 클라이언트 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 `send` 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *`ITEMS_COUNT`*를 반환할 최대 템플릿 수로 바꿉니다. 값은 최소 1에서 최대 10이어야 합니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. Amazon SES가 템플릿 목록을 반환합니다.

```
node ses_listtemplates.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

이메일 템플릿 가져오기

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용할 이메일 템플릿을 가져옵니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. *### ## ####* 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_gettemplate.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SES 클라이언트 클래스의 `GetTemplateCommand` 메서드에 대한 `TemplateName` 파라미터를 전달할 객체를 생성합니다. `GetTemplateCommand` 메서드를 직접적으로 호출하려면 Amazon SES 클라이언트 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 `send` 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *TEMPLATE_NAME*을 반환할 템플릿의 이름으로 바꿉니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. Amazon SES가 템플릿 세부 정보를 반환합니다.

```
node ses_gettemplate.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

이메일 템플릿 생성

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용할 이메일 템플릿을 생성합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. `### ## ####` 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_createtemplate.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

`TemplateName`, `HtmlPart`, `SubjectPart` 및 `TextPart`를 포함하여 SES 클라이언트 클래스의 `CreateTemplateCommand` 메서드에 대한 파라미터를 전달할 객체를 생성합니다. `CreateTemplateCommand` 메서드를 직접적으로 호출하려면 Amazon SES 클라이언트 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 send 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 메서드를 async/await 패턴으로 사용합니다. send 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *TEMPLATE_NAME*을 새 템플릿의 이름으로, *HTML_CONTENT*를 이메일의 HTML 태그 지정 콘텐츠로, *SUBJECT*를 이메일 제목으로, *TEXT_CONTENT*를 이메일의 텍스트로 바꿉니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. 템플릿이 Amazon SES에 추가됩니다.

```
node ses_createtemplate.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

이메일 템플릿 업데이트

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용할 이메일 템플릿을 생성합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. ### ## #### 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_updatetemplate.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SES 클라이언트 클래스의 `UpdateTemplateCommand` 메서드에 전달된 필수 `TemplateName` 파라미터와 함께 템플릿에서 업데이트하려는 `Template` 파라미터 값을 전달할 객체를 생성합니다. `UpdateTemplateCommand` 메서드를 직접적으로 호출하려면 Amazon SES 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 `send` 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *`TEMPLATE_NAME`*을 템플릿 이름으로, *`HTML_CONTENT`*를 이메일의 HTML 태그 지정 콘텐츠로, *`SUBJECT`*를 이메일 제목으로, *`TEXT_CONTENT`*를 이메일의 텍스트로 바꿉니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. Amazon SES가 템플릿 세부 정보를 반환합니다.

```
node ses_updatetemplate.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

이메일 템플릿 삭제

이 예에서는 Node.js 모듈을 사용하여 Amazon SES에 사용할 이메일 템플릿을 생성합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. `### ## ####` 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_deletetemplate.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SES 클라이언트 클래스의 `DeleteTemplateCommand` 메서드에 필수 `TemplateName` 파라미터를 전달할 객체를 생성합니다. `DeleteTemplateCommand` 메서드를 직접적으로 호출하려면 Amazon SES 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 `send` 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *TEMPLATE_NAME*을 삭제할 템플릿의 이름으로 바꿉니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. Amazon SES가 템플릿 세부 정보를 반환합니다.

```
node ses_deletetemplate.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

Amazon SES를 사용하여 이메일 전송

이 Node.js 코드 예제는 다음을 보여 줍니다.

- 테스트 또는 HTML 이메일을 전송합니다.
- 이메일 템플릿을 기반으로 이메일을 전송합니다.
- 이메일 템플릿을 기반으로 대량 이메일을 전송합니다.

Amazon SES API에서는 이메일 메시지 작성에 대해 원하는 제어 정도에 따라 서식 지정 및 원시라는 두 가지 이메일 전송 방법을 선택할 수 있습니다. 자세한 내용은 Amazon SES API를 사용하여 서식이 지정된 이메일 보내기 및 Amazon SES API를 사용하여 원시 이메일 보내기 단원을 참조하세요.

시나리오

이 예제에서는 일련의 Node.js 모듈을 사용하여 다양한 방법으로 이메일을 전송합니다. Node.js 모듈은 SDK JavaScript Form을 사용하여 SES 클라이언트 클래스의 다음 메서드를 사용하여 이메일 템플릿을 만들고 사용합니다.

- `SendEmailCommand`
- `SendTemplatedEmailCommand`
- `SendBulkTemplatedEmailCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 Node TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. 의 지침을 따르십시오 GitHub.

- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.
>
> - 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> - CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

이메일 메시지 전송 요구 사항

Amazon SES에서는 이메일 메시지를 작성하는 즉시 전송 대기열에 넣습니다. `SendEmailCommand` 메서드를 사용하여 이메일을 전송하려면 메시지는 다음 요구 사항을 충족해야 합니다.

- 확인된 이메일 주소 또는 도메인에서 메시지를 전송해야 합니다. 확인되지 않은 주소 또는 도메인을 사용하여 이메일을 전송하려고 시도하면 작업 결과로 `"Email address not verified"` 오류가 발생합니다.
- 계정이 여전히 Amazon SES 샌드박스에 있는 경우 확인된 주소 또는 도메인으로만 또는 Amazon SES 메일박스 시뮬레이터와 연결된 이메일 주소로만 전송할 수 있습니다. 자세한 내용은 Amazon Simple Email Service 개발자 안내서의 Amazon SES에서 확인된 자격 증명 단원을 참조하세요.
- 첨부 파일을 포함한 메시지의 총 크기는 10MB 미만이어야 합니다.
- 메시지에 최소 하나 이상의 수신자 이메일 주소가 포함되어야 합니다. 수신자 주소는 받는 사람: 주소, 참조: 주소 또는 숨은 참조: 주소일 수 있습니다. 수신자 이메일 주소가 유효하지 않은 경우(즉, `UserName@[SubDomain.]Domain.TopLevelDomain` 형식이 아닌 경우) 메시지에 유효한 다른 수신자가 포함되어 있더라도 전체 메시지가 거부됩니다.
- 메시지에는 받는 사람:, 참조: 및 숨은 참조: 필드 전체에서 50명을 초과하는 수신자가 포함될 수 없습니다. 더 많은 대상에게 이메일 메시지를 전송해야 하는 경우 수신자 목록을 50명 이하의 여러 그룹으로 나눈 다음 `sendEmail` 메서드를 여러 번 호출하여 각 그룹에게 메시지를 전송할 수 있습니다.

이메일 전송

이 예제에서는 Node.js 모듈을 사용하여 Amazon SES에서 이메일을 전송합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. *### ## ####* 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_sendemail.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

발신자 및 수신자 주소, 제목, 일반 텍스트 및 HTML 형식의 이메일 본문을 포함하여 전송할 이메일을 정의하는 파라미터 값을 SES 클라이언트 클래스의 `SendEmailCommand` 메서드에 전달할 객체를 생성합니다. `SendEmailCommand` 메서드를 직접적으로 호출하려면 Amazon SES 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 `send` 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *RECEIVER_ADDRESS*를 이메일 수신 주소로 바꾸고, *SENDER_ADDRESS*를 이메일 발신 주소로 바꿉니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. 이메일이 Amazon SES에서 전송하기 위해 대기됩니다.

```
node ses_sendemail.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

템플릿을 사용한 이메일 전송

이 예제에서는 Node.js 모듈을 사용하여 Amazon SES에서 이메일을 전송합니다. 파일 이름이 `ses_sendtemplatedemail.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

발신자 및 수신자 주소, 제목, 일반 텍스트 및 HTML 형식의 이메일 본문을 포함하여 전송할 이메일을 정의하는 파라미터 값을 SES 클라이언트 클래스의 `SendTemplatedEmailCommand` 메서드에 전달할 객체를 생성합니다. `SendTemplatedEmailCommand` 메서드를 직접적으로 호출하려면 Amazon SES 클라이언트 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 `send` 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *### ###### ###, RECEIVER_ADDRESS# #### ## ###, SENDER_ADDRESS# #### ## ### ###, TEMPLATE_NAME# ### #### #####. 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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. 이메일이 Amazon SES에서 전송하기 위해 대기됩니다.

```
node ses_sendtemplatedemail.js
```

이 예제 코드는 여기에서 확인할 수 있습니다. GitHub

템플릿을 사용한 대량 이메일 전송

이 예제에서는 Node.js 모듈을 사용하여 Amazon SES에서 이메일을 전송합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `sesClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. `### ## ####` 바꾸세요 AWS .

```
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 };
```

이 예제 코드는 여기에서 찾을 수 있습니다 GitHub.

파일 이름이 `ses_sendbulktemplatedemail.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

발신자 및 수신자 주소, 제목, 일반 텍스트 및 HTML 형식의 이메일 본문을 포함하여 전송할 이메일을 정의하는 파라미터 값을 SES 클라이언트 클래스의 `SendBulkTemplatedEmailCommand` 메서드에 전달할 객체를 생성합니다. `SendBulkTemplatedEmailCommand` 메서드를 직접적으로 호출하려면 Amazon SES 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

> **Note**
>
> 이 예제에서는 필수 AWS Service V3 패키지 클라이언트, V3 명령을 가져와서 사용하고 send 메서드를 async/await 패턴으로 사용합니다. 대신 몇 가지 사소한 변경을 통해 V2 명령을 사용하여 이 예를 생성할 수 있습니다. 자세한 내용은 V3 명령 사용 섹션을 참조하세요.

> **Note**
>
> *RECEIVER_ADDRESSES*를 이메일 수신 주소로 바꾸고, *SENDER_ADDRESS*를 이메일 발신 주소로 바꿉니다.

```
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;
  }
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다. 이메일이 Amazon SES에서 전송하기 위해 대기됩니다.

```
node ses_sendbulktemplatedemail.js
```

이 예제 코드는 여기에서 찾을 수 있습니다. GitHub

Amazon Simple Notification Service 예

Amazon Simple Notification Service(Amazon SNS)는 구독 중인 엔드포인트 또는 클라이언트에 대한 메시지 전달 또는 전송을 조정 및 관리하는 웹 서비스입니다.

Amazon SNS에는 게시자와 구독자 또는 생산자와 소비자라고 하는 두 가지 클라이언트 유형이 있습니다.

게시자는 주제에 대한 메시지를 생산 및 발송함으로써 구독자와 비동시적으로 통신하는 논리적 액세스 및 커뮤니케이션 채널입니다. 구독자(웹 서버, 이메일 주소, Amazon SQS 대기열, AWS Lambda 함수)는 주제를 구독할 때 지원되는 프로토콜(Amazon SQS, HTTP/S, 이메일, SMS, AWS Lambda) 중 하나를 통해 메시지 또는 알림을 소비하거나 수신합니다.

Amazon SNS용 JavaScript API는 Class: SNS를 통해 노출됩니다.

주제

- Amazon SNS에서 주제 관리
- Amazon SNS에서 메시지 게시
- Amazon SNS에서 구독 관리
- Amazon SNS를 통한 SMS 메시지 전송

Amazon SNS에서 주제 관리

이 Node.js 코드 예제는 다음을 보여 줍니다.

- Amazon SNS에서 알림을 게시할 수 있는 주제를 생성하는 방법
- Amazon SNS에서 생성된 주제를 삭제하는 방법
- 사용 가능한 주제 목록을 가져오는 방법.
- 주제 속성을 가져오고 설정하는 방법.

시나리오

이 예에서는 일련의 Node.js 모듈을 사용하여 Amazon SNS 주제를 생성, 나열 및 삭제하고 주제 속성을 처리합니다. 이 Node.js 모듈은 SDK for JavaScript에서 SNS 클라이언트 클래스의 다음 메서드를 사용하여 주제를 관리합니다.

- `CreateTopicCommand`
- `ListTopicsCommand`
- `DeleteTopicCommand`
- `GetTopicAttributesCommand`
- `SetTopicAttributesCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.
>
> - 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> - CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

주제 생성

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS 주제를 생성합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `create-topic.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SNS 클라이언트 클래스의 `CreateTopicCommand` 메서드에 새 주제의 `Name`을 전달할 객체를 생성합니다. `CreateTopicCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다. 반환되는 `data`에는 주제의 ARN이 포함됩니다.

> **Note**
>
> *TOPIC_NAME*을 주제 이름으로 바꿉니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node create-topic.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

주제 나열

이 예에서는 Node.js 모듈을 사용하여 모든 Amazon SNS 주제를 나열합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `list-topics.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SNS 클라이언트 클래스의 `ListTopicsCommand` 메서드에 전달할 비어 있는 객체를 생성합니다. `ListTopicsCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 서비스 객체를 간접적으로 호

출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다. 반환된 data에는 주제 Amazon 리소스 이름(ARN)의 배열이 포함되어 있습니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node list-topics.js
```

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

주제 삭제

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS 주제를 삭제합니다.

libs 디렉터리를 생성하고 파일 이름이 snsClient.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `delete-topic.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SNS 클라이언트 클래스의 `DeleteTopicCommand` 메서드에 전달할 삭제할 주제의 `TopicArn`을 포함하는 객체를 생성합니다. `DeleteTopicCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *TOPIC_ARN*을 삭제하려는 주제의 Amazon 리소스 이름(ARN)으로 바꿉니다.

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

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node delete-topic.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

주제 속성 가져오기

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS 주제의 속성을 검색합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `get-topic-attributes.js`인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

SNS 클라이언트 클래스의 `GetTopicAttributesCommand` 메서드에 전달할 삭제할 주제의 `TopicArn`을 포함하는 객체를 생성합니다. `GetTopicAttributesCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하여 파라미터 객체를 전달합니다.

> **Note**
>
> *TOPIC_ARN*을 주제의 ARN으로 바꿉니다.

```
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: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx: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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node get-topic-attributes.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

주제 속성 설정

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS 주제의 변경 가능한 속성을 설정합니다.

libs 디렉터리를 생성하고 파일 이름이 snsClient.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 set-topic-attributes.js인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

속성을 설정하려고 하는 주제의 TopicArn, 설정할 속성의 이름, 해당 속성의 새 값을 포함하여 속성 업데이트를 위한 파라미터를 포함하는 객체를 생성합니다. Policy, DisplayName 및 DeliveryPolicy 속성만 설정할 수 있습니다. SNS 클라이언트 클래스의 SetTopicAttributesCommand 메서드에 파라미터를 전달합니다. SetTopicAttributesCommand 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *ATTRIBUTE_NAME*을 설정할 속성의 이름으로, *TOPIC_ARN*을 속성을 설정하려는 주제의 Amazon 리소스 이름(ARN)으로, *NEW_ATTRIBUTE_VALUE*를 해당 속성의 새 값으로 바꿉니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node set-topic-attributes.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

Amazon SNS에서 메시지 게시

이 Node.js 코드 예제는 다음을 보여 줍니다.

- Amazon SNS 주제에 메시지를 게시하는 방법

시나리오

이 예에서는 일련의 Node.js 모듈을 사용하여 Amazon SNS의 메시지를 주제 엔드포인트, 이메일 또는 전화번호에 게시합니다. 이 Node.js 모듈은 SDK for JavaScript에서 SNS 클라이언트 클래스의 다음 메서드를 사용하여 메시지를 전송합니다.

- PublishCommand

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.
>
> - 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> - CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

SNS 주제에 메시지 게시

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS 주제에 메시지를 게시합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `publish-topic.js`인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

메시지 텍스트와 Amazon SNS 주제의 Amazon 리소스 이름(ARN)을 비롯하여 메시지 게시를 위한 파라미터가 포함된 객체를 생성합니다. 사용 가능한 SMS 속성에 대한 세부 정보는 SetSMSAttributes를 참조하십시오.

SNS 클라이언트 클래스의 PublishCommand 메서드에 파라미터를 전달합니다. Amazon SNS 클라이언트 서비스 객체를 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *MESSAGE_TEXT*를 메시지 텍스트로 바꾸고, *TOPIC_ARN*을 SNS 주제의 ARN으로 바꿉니다.

```
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-xxxxxxxxxxxx'
  // }
  return response;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node publish-topic.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

Amazon SNS에서 구독 관리

이 Node.js 코드 예제는 다음을 보여 줍니다.

- Amazon SNS 주제의 모든 구독을 나열하는 방법
- 이메일 주소, 애플리케이션 엔드포인트 또는 AWS Lambda 함수에서 Amazon SNS 주제를 구독하는 방법
- Amazon SNS 주제의 구독을 취소하는 방법

시나리오

이 예에서는 일련의 Node.js 모듈을 사용하여 Amazon SNS 주제에 알림 메시지를 게시합니다. 이 Node.js 모듈은 SDK for JavaScript에서 SNS 클라이언트 클래스의 다음 메서드를 사용하여 주제를 관리합니다.

- `ListSubscriptionsByTopicCommand`
- `SubscribeCommand`
- `ConfirmSubscriptionCommand`
- `UnsubscribeCommand`

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.
>
> - 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> - CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

주제에 대한 구독 나열

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS 주제에 대한 모든 구독을 나열합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `list-subscriptions-by-topic.js`인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

구독을 나열할 주제에 대한 `TopicArn` 파라미터를 포함하는 객체를 생성합니다. `SNS` 클라이언트 클래스의 `ListSubscriptionsByTopicCommand` 메서드에 파라미터를 전달합니다. `ListSubscriptionsByTopicCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *TOPIC_ARN*을 구독을 나열하려는 주제의 Amazon 리소스 이름(ARN)으로 바꿉니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node list-subscriptions-by-topic.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

이메일 주소에서 주제 구독

이 예에서는 Node.js 모듈을 사용하여 이메일 주소에서 Amazon SNS 주제의 SMTP 이메일 메시지를 수신하도록 이메일 주소에서 주제를 구독합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `subscribe-email.js`인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

`email` 프로토콜, 구독할 주제의 `TopicArn`, 메시지 `Endpoint`로 사용되는 이메일 주소를 지정하기 위한 `Protocol` 파라미터를 포함하는 객체를 생성합니다. SNS 클라이언트 클래스의 `SubscribeCommand` 메서드에 파라미터를 전달합니다. 이 항목의 다른 예에 나와 있듯이, `subscribe` 메서드를 사용하면 전달된 파라미터에 사용되는 값에 따라 여러 다양한 엔드포인트에서 Amazon SNS 주제를 구독할 수 있습니다.

`SubscribeCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *`TOPIC_ARN`*을 주제의 Amazon 리소스 이름(ARN)으로 바꾸고, *`EMAIL_ADDRESS`*를 구독할 이메일 주소로 바꿉니다.

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

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node subscribe-email.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

구독 확인

이 예에서는 Node.js 모듈을 사용하여 이전 구독 작업에서 엔드포인트로 전송한 토큰을 검증함으로써 엔드포인트 소유자의 이메일 수신 의도를 확인합니다.

libs 디렉터리를 생성하고 파일 이름이 snsClient.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

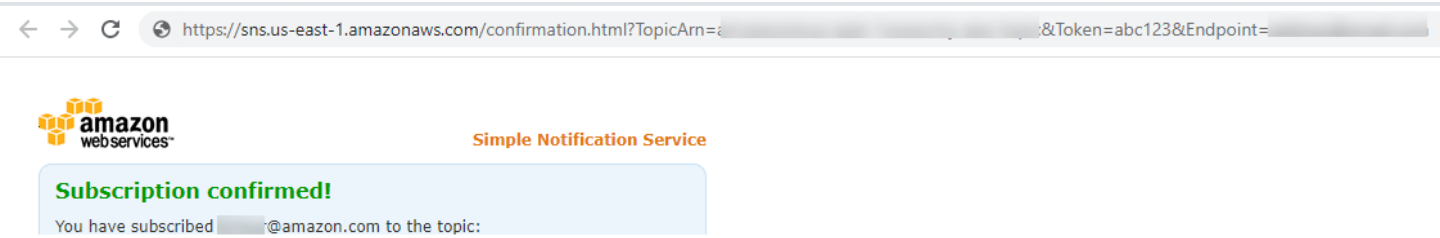
```
import { SNSClient } from "@aws-sdk/client-sns";

// 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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 confirm-subscription.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

TOPIC_ARN 및 TOKEN을 포함한 파라미터를 정의하고, AuthenticateOnUnsubscribe에 대해 TRUE 또는 FALSE 값을 정의합니다.

토큰은 이전 SUBSCRIBE 작업 중에 엔드포인트 소유자에게 전송된 수명이 짧은 토큰입니다. 예를 들어 이메일 엔드포인트의 경우 TOKEN은 이메일 소유자에게 전송된 구독 확인 이메일의 URL에 있습니다. 예를 들어 abc123은 다음 URL의 토큰입니다.



ConfirmSubscriptionCommand 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *TOPIC_ARN*을 주제의 Amazon 리소스 이름(ARN)으로 바꾸고, *TOKEN*을 이전 Subscribe 작업에서 엔드포인트 소유자에게 전송한 URL의 토큰 값으로 바꾸고 *AuthenticateOnUnsubscribe*를 정의합니다. TRUE 또는 FALSE 값을 사용합니다.

```
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:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node confirm-subscription.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

애플리케이션 엔드포인트에서 주제 구독

이 예에서는 Node.js 모듈을 사용하여 모바일 애플리케이션 엔드포인트에서 Amazon SNS 주제의 알림을 수신하도록 모바일 애플리케이션 엔드포인트에서 주제를 구독합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `subscribe-app.js`인 Node.js 모듈을 생성합니다. 필수 모듈 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

`Protocol` 파라미터가 포함된 객체를 생성하여 `application` 프로토콜, 구독할 주제의 `TopicArn`, `Endpoint` 파라미터에 대한 모바일 애플리케이션 엔드포인트의 Amazon 리소스 이름(ARN)을 지정합니다. SNS 클라이언트 클래스의 `SubscribeCommand` 메서드에 파라미터를 전달합니다.

`SubscribeCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *`TOPIC_ARN`*을 주제의 Amazon 리소스 이름(ARN)으로 바꾸고, *`MOBILE_ENDPOINT_ARN`*을 주제를 구독하는 엔드포인트로 바꿉니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node subscribe-app.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

Lambda 함수에서 주제 구독

이 예에서는 Node.js 모듈을 사용하여 AWS Lambda 함수에서 Amazon SNS 주제의 알림을 수신하도록 Lambda 함수에서 주제를 구독합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `subscribe-lambda.js`인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

`Protocol` 파라미터가 포함된 객체를 생성하여 `lambda` 프로토콜, 구독할 주제의 `TopicArn`, AWS Lambda 함수의 Amazon 리소스 이름(ARN)을 `Endpoint` 파라미터로 지정합니다. `SNS` 클라이언트 클래스의 `SubscribeCommand` 메서드에 파라미터를 전달합니다.

`SubscribeCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *TOPIC_ARN*을 주제의 Amazon 리소스 이름(ARN)으로 바꾸고, *LAMBDA_FUNCTION_ARN*을 Lambda 함수의 Amazon 리소스 이름(ARN)으로 바꿉니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node subscribe-lambda.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

주제의 구독 취소

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS 주제 구독을 취소합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 unsubscribe.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SubscriptionArn 파라미터가 포함된 객체를 생성하여 취소할 구독의 Amazon 리소스 이름(ARN)을 지정합니다. SNS 클라이언트 클래스의 UnsubscribeCommand 메서드에 파라미터를 전달합니다.

UnsubscribeCommand 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *TOPIC_SUBSCRIPTION_ARN*을 취소할 구독의 Amazon 리소스 이름(ARN)으로 바꿉니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node unsubscribe.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

Amazon SNS를 통한 SMS 메시지 전송

이 Node.js 코드 예제는 다음을 보여 줍니다.

- Amazon SNS에 대한 SMS 메시징 기본 설정을 가져오고 설정하는 방법
- 전화번호를 점검하여 SMS 메시지 수신을 옵트아웃했는지 여부를 확인하는 방법.
- SMS 메시지 수신을 옵트아웃한 전화번호의 목록을 가져오는 방법.
- SMS 메시지를 전송하는 방법.

시나리오

사용자는 Amazon SNS를 사용하여 SMS 수신 가능한 디바이스에 문자 메시지 또는 SMS 메시지를 전송할 수 있습니다. 전화번호로 메시지를 직접 전송할 수 있으며, 전화번호에서 주제를 구독하고 메시지를 주제로 전송하여 메시지를 여러 전화번호로 한 번에 전송할 수 있습니다.

이 예에서는 일련의 Node.js 모듈을 사용하여 Amazon SNS의 SMS 문자 메시지를 SMS 지원 디바이스에 게시합니다. 이 Node.js 모듈은 SDK for JavaScript에서 SNS 클라이언트 클래스의 다음 메서드를 사용하여 SMS 메시지를 게시합니다.

- GetSMSAttributesCommand
- SetSMSAttributesCommand
- CheckIfPhoneNumberIsOptedOutCommand
- ListPhoneNumbersOptedOutCommand
- PublishCommand

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.
>
> - 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> - CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

SMS 속성 가져오기

Amazon SNS를 사용하여 전송을 최적화하는 방법(비용 또는 안정성 있는 전송), 월 지출 한도, 메시지 전송을 로깅하는 방법, 일일 SMS 사용 보고서를 구독하는지 여부 등 SMS 메시징에 대한 기본 설정을 지정합니다. 이러한 기본 설정을 검색하여 Amazon SNS의 SMS 속성으로 설정합니다.

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS에서 현재 SMS 속성을 가져옵니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `get-sms-attributes.js`인 Node.js 모듈을 생성합니다.

필수 클라이언트 및 패키지 다운로드를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다. 가져올 개별 속성의 이름을 포함하여 SMS 속성을 가져오기 위한 파라미터를 포함하는 객체를 생성합니다. 사용 가능한 SMS 속성에 대한 자세한 내용은 Amazon Simple Notification Service API 참조의 SetSMSAttributes를 참조하세요.

이 예제에서는 SMS 메시지를 최저 비용이 발생하도록 메시지 전송을 최적화하는 `Promotional`로 전송할지 또는 최고 안정성을 달성하도록 메시지 전송을 최적화하는 `Transactional`로 전송할지를 제어하는 `DefaultSMSType` 속성을 가져옵니다. SNS 클라이언트 클래스의 `SetTopicAttributesCommand` 메서드에 파라미터를 전달합니다. `SetSMSAttributesCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *ATTRIBUTE_NAME*을 속성 이름으로 바꿉니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node get-sms-attributes.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

SMS 속성 설정

이 예에서는 Node.js 모듈을 사용하여 Amazon SNS에서 현재 SMS 속성을 가져옵니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `set-sms-attribute-type.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다. 설정할 개별 속성의 이름과 각 속성에 설정할 값을 포함하여 SMS 속성을 설정하기 위한 파라미터를 포함하는 객체를 생성합니다. 사용 가능한 SMS 속성에 대한 자세한 내용은 Amazon Simple Notification Service API 참조의 SetSMSAttributes를 참조하세요.

다음 예제에서는 `DefaultSMSType` 속성을 `Transactional`로 설정하여 최고의 안정성을 달성하도록 메시지 전송을 최적화합니다. SNS 클라이언트 클래스의 `SetTopicAttributesCommand` 메서드에 파라미터를 전달합니다. `SetSMSAttributesCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node set-sms-attribute-type.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

전화번호가 옵트아웃되었는지 여부 확인

이 예제에서는 Node.js 모듈을 사용하여 전화 번호가 SMS 메시지 수신에서 옵트아웃되었는지 여부를 확인합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `check-if-phone-number-is-opted-out.js`인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다. 파라미터로 확인할 전화번호를 포함하는 객체를 생성합니다.

이 예제에서는 확인할 전화번호를 지정하는 `PhoneNumber` 파라미터를 설정합니다. SNS 클라이언트 클래스의 `CheckIfPhoneNumberIsOptedOutCommand` 메서드에 객체를 전달합니다. `CheckIfPhoneNumberIsOptedOutCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> 1.
>
> *PHONE_NUMBER*를 전화번호로 바꿉니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node check-if-phone-number-is-opted-out.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

옵트아웃된 전화번호 나열

이 예제에서는 Node.js 모듈을 사용하여 SMS 메시지 수신에서 옵트아웃한 전화번호의 목록을 가져옵니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `list-phone-numbers-opted-out.js`인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다. 비어 있는 객체를 파라미터로 생성합니다.

SNS 클라이언트 클래스의 `ListPhoneNumbersOptedOutCommand` 메서드에 객체를 전달합니다. `ListPhoneNumbersOptedOutCommand` 메서드를 직접적으로 호출하려면 Amazon SNS 클라이언트 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

```
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;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node list-phone-numbers-opted-out.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

SMS 메시지 게시

이 예제에서는 Node.js 모듈을 사용하여 SMS 메시지를 전화번호에 전송합니다.

`libs` 디렉터리를 생성하고 파일 이름이 `snsClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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({});
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 publish-sms.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다. Message 및 PhoneNumber 파라미터를 포함하는 객체를 생성합니다.

SMS 메시지를 전송할 때 E.164 형식을 사용하여 전화번호를 지정합니다. E.164는 국제 통신에 사용되는 전화번호 구조의 표준입니다. 이 형식을 따르는 전화번호는 최대 15자리 숫자를 사용할 수 있으며 더하기 문자(+) 및 국가 코드가 접두사로 추가됩니다. 예를 들어, E.164 형식의 미국 전화번호는 +1001XXX5550100으로 표시될 수 있습니다.

이 예제에서는 메시지를 전송할 전화번호를 지정하는 PhoneNumber 파라미터를 설정합니다. SNS 클라이언트 클래스의 PublishCommand 메서드에 객체를 전달합니다. PublishCommand 메서드를 직접적으로 호출하려면 Amazon SNS 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다.

> **Note**
>
> *TEXT_MESSAGE*를 텍스트 메시지로 바꾸고, *PHONE_NUMBER*를 전화번호로 바꿉니다.

```
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-xxxxxxxxxxxx'
  // }
  return response;
};
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node publish-sms.js
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

Amazon Transcribe 예

Amazon Transcribe를 사용하면 개발자가 애플리케이션에 음성 텍스트 변환 기능을 쉽게 추가할 수 있습니다.

Amazon Transcribe용 JavaScript API는 TranscribeService 클라이언트 클래스를 통해 노출됩니다.

주제

- Amazon Transcribe 예
- Amazon Transcribe Medical 예

Amazon Transcribe 예

이 예에서는 일련의 Node.js 모듈에서 `TranscribeService` 클라이언트 클래스의 다음 메서드를 사용하여 트랜스크립션 작업을 생성, 나열 및 삭제합니다.

- `StartTranscriptionJobCommand`
- `ListTranscriptionJobsCommand`
- `DeleteTranscriptionJobCommand`

Amazon Transcribe 사용자에 관한 자세한 내용은 Amazon Transcribe 개발자 안내서를 참조하세요.

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.
>
> - 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> - CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

Amazon Transcribe 작업 시작

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Transcribe 트랜스크립션 작업을 시작하는 방법을 보여줍니다. 자세한 내용은 StartTranscriptionJobCommand를 참조하세요.

libs 디렉터리를 생성하고 파일 이름이 transcribeClient.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 transcribe-create-job.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 파라미터 객체를 생성하여 필수 파라미터를 지정합니다. StartMedicalTranscriptionJobCommand 명령을 사용하여 작업을 시작합니다.

> **Note**
>
> *MEDICAL_JOB_NAME*을 트랜스크립션 작업 이름으로 바꿉니다. *OUTPUT_BUCKET_NAME*에 출력이 저장되는 Amazon S3 버킷을 지정합니다. *JOB_TYPE*에 작업 유형을 지정합니다. *SOURCE_LOCATION*에 소스 파일의 위치를 지정합니다. *SOURCE_FILE_LOCATION*에 입력 미디어 파일의 위치를 지정합니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node transcribe-create-job.js
```

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Transcribe 작업 나열

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Transcribe 트랜스크립션 작업을 나열하는 방법을 보여줍니다. 수정할 수 있는 다른 설정에 관한 자세한 내용은 `ListTranscriptionJobCommand` 단원을 참조하세요.

`libs` 디렉터리를 생성하고 파일 이름이 `transcribeClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `transcribe-list-jobs.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 필수 파라미터를 사용하여 파라미터 객체를 생성합니다.

> **Note**
>
> *KEY_WORD*를 반환된 작업 이름에 포함해야 하는 키워드로 바꿉니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node transcribe-list-jobs.js
```

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Transcribe 작업 삭제

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Transcribe 트랜스크립션 작업을 삭제하는 방법을 보여줍니다. 옵션에 관한 자세한 내용은 `DeleteTranscriptionJobCommand` 단원을 참조하세요.

`libs` 디렉터리를 생성하고 파일 이름이 `transcribeClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `transcribe-delete-job.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. AWS 리전과 삭제하려는 작업 이름을 지정합니다.

> **Note**
>
> *`JOB_NAME`*을 삭제할 작업의 이름으로 바꿉니다.

```
// 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, 'transciption_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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node transcribe-delete-job.js
```

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Transcribe Medical 예

이 예에서는 일련의 Node.js 모듈에서 `TranscribeService` 클라이언트 클래스의 다음 메서드를 사용하여 의료 트랜스크립션 작업을 생성, 나열 및 삭제합니다.

- `StartMedicalTranscriptionJobCommand`
- `ListMedicalTranscriptionJobsCommand`
- `DeleteMedicalTranscriptionJobCommand`

Amazon Transcribe 사용자에 관한 자세한 내용은 Amazon Transcribe 개발자 안내서를 참조하세요.

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

Important

이 예는 ECMAScript6(ES6)를 사용하여 클라이언트 서비스 객체 및 명령을 가져오거나 내보내는 방법을 보여줍니다.

- 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
- CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

Amazon Transcribe Medical 트랜스크립션 작업 시작

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Transcribe Medical 트랜스크립션 작업을 시작하는 방법을 보여줍니다. 자세한 내용은 startMedicalTranscriptionJob 단원을 참조하세요.

`libs` 디렉터리를 생성하고 파일 이름이 `transcribeClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `transcribe-create-medical-job.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 파라미터 객체를 생성하여 필수 파라미터를 지정합니다. `StartMedicalTranscriptionJobCommand` 명령을 사용하여 의료 작업을 시작합니다.

Note

*MEDICAL_JOB_NAME*을 의료 트랜스크립션 작업 이름으로 바꿉니다. *OUTPUT_BUCKET_NAME*에 출력이 저장되는 Amazon S3 버킷을 지정합니다. *JOB_TYPE*에 작업 유형을 지정합니다. *SOURCE_LOCATION*에 소스 파일의 위치를 지정합니다. *SOURCE_FILE_LOCATION*에 입력 미디어 파일의 위치를 지정합니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node transcribe-create-medical-job.js
```

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Transcribe Medical 작업 나열

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Transcribe 트랜스크립션 작업을 나열하는 방법을 보여줍니다. 자세한 내용은 ListTranscriptionMedicalJobsCommand 단원을 참조하세요.

`libs` 디렉터리를 생성하고 파일 이름이 `transcribeClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. *REGION*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `transcribe-list-medical-jobs.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 필수 파라미터를 사용하여 파라미터 객체를 생성하고 `ListMedicalTranscriptionJobsCommand` 명령을 사용하여 의료 작업을 나열합니다.

> **Note**
>
> *KEYWORD*를 반환된 작업 이름에 포함해야 하는 키워드로 바꿉니다.

```
// 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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node transcribe-list-medical-jobs.js
```

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon Transcribe Medical 작업 삭제

이 예에서는 AWS SDK for JavaScript를 사용하여 Amazon Transcribe 트랜스크립션 작업을 삭제하는 방법을 보여줍니다. 옵션에 관한 자세한 내용은 `DeleteTranscriptionMedicalJobCommand` 단원을 참조하세요.

`libs` 디렉터리를 생성하고 파일 이름이 `transcribeClient.js`인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. *`REGION`*을 AWS 리전으로 바꿉니다.

```
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 };
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

파일 이름이 `transcribe-delete-job.js`인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. 필수 파라미터를 사용하여 파라미터 객체를 생성하고 `DeleteMedicalJobCommand` 명령을 사용하여 의료 작업을 삭제합니다.

> **Note**
>
> *JOB_NAME*을 삭제할 작업의 이름으로 바꿉니다.

```
// 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_transciption_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();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
node transcribe-delete-medical-job.js
```

이 샘플 코드는 GitHub에서 찾을 수 있습니다.

Amazon EC2 인스턴스에서 Node.js 설정

Node.js 를 SDK와 함께 사용하는 일반적인 시나리오는 Amazon Elastic Compute Cloud (Amazon EC2) 인스턴스에서 Node.js 웹 애플리케이션을 설정하고 실행하는 것입니다. JavaScript 이 자습서에

서는 Linux 인스턴스를 생성하고, SSH를 사용하여 해당 인스턴스에 연결한 다음, 해당 인스턴스에서 실행할 Node.js를 설치합니다.

필수 조건

이 자습서에서는 인터넷에서 접근 가능하고 SSH를 사용하여 연결할 수 있는 퍼블릭 DNS 이름으로 Linux 인스턴스를 이미 시작했다고 가정합니다. 자세한 내용은 Amazon EC2 Linux 인스턴스용 사용 설명서의 1단계: 인스턴스 시작 단원을 참조하세요.

> **Important**
>
> 새 Amazon EC2 인스턴스를 시작할 때 Amazon Linux 2023 Amazon Machine Image(AMI)를 사용합니다.

보안 그룹이 SSH(포트 22), HTTP(포트 80), HTTPS(포트 443) 연결을 허용하도록 구성되어야 합니다. 이러한 사전 조건에 관한 자세한 내용은 Amazon EC2 Linux 인스턴스용 사용 설명서의 Amazon EC2 사용 설정 단원을 참조하세요.

절차

다음 절차는 Amazon Linux 인스턴스에서 Node.js를 설치하는 데 도움이 됩니다. 이 서버를 사용하여 Node.js 웹 애플리케이션을 호스팅할 수 있습니다.

Linux 인스턴스에서 Node.js를 설정하려면

1. SSH를 사용하여 `ec2-user`로 Linux 인스턴스에 연결합니다.
2. 명령줄에 다음을 입력하여 노드 버전 관리자(`nvm`)를 설치합니다.

   > **Warning**
   >
   > AWS 다음 코드는 제어하지 않습니다. 실행하기 전에 먼저 신뢰성과 무결성을 확인해야 합니다. 이 코드에 대한 자세한 내용은 nvm GitHub 저장소에서 찾을 수 있습니다.

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
   ```

   `nvm`을 사용하면 여러 버전의 Node.js를 설치할 수 있고 여러 버전 간을 전환할 수 있기 때문에 여기서는 `nvm`을 사용하여 Node.js를 설치합니다.

3. 명령줄에 다음을 nvm 입력하여 로드합니다.

```
source ~/.bashrc
```

4. 명령줄에 다음을 입력하여 nvm을 사용해 최신 LTS 버전의 Node.js를 설치합니다.

```
nvm install --lts
```

Node.js를 설치하면 노드 패키지 관리자(npm)도 설치되므로 필요에 따라 추가 모듈을 설치할 수 있습니다.

5. 명령줄에 다음을 입력하여 Node.js가 올바르게 설치되고 실행되는지 테스트합니다.

```
node -e "console.log('Running Node.js ' + process.version)"
```

이렇게 하면 실행 중인 Node.js의 버전을 보여 주는 메시지가 다음과 같이 표시됩니다.

```
Running Node.js VERSION
```

> **Note**
>
> 노드 설치는 현재 Amazon EC2 세션에만 적용됩니다. CLI 세션을 다시 시작하는 경우 nvm을 다시 사용하여 설치된 노드 버전을 활성화해야 합니다. 인스턴스가 종료되면 노드를 다시 설치해야 합니다. 이에 대한 대안은 다음 항목에 설명된 대로 유지하려는 구성이 있다면 Amazon EC2 인스턴스의 Amazon Machine Image(AMI)를 만드는 것입니다.

Amazon Machine Image(AMI) 생성

Amazon EC2 인스턴스에 Node.js를 설치한 후에는 해당 인스턴스에서 Amazon Machine Image(AMI)를 생성할 수 있습니다. AMI를 생성하면 동일한 Node.js 설치에서 여러 Amazon EC2 인스턴스를 쉽게 프로비저닝할 수 있습니다. 기존 인스턴스에서 AMI를 생성하는 방법에 관한 자세한 내용은 Amazon EC2 Linux 인스턴스용 사용 설명서의 Amazon EBS-backed Linux AMI 생성 단원을 참조하세요.

관련 리소스

이 주제에 사용되는 명령과 소프트웨어에 관한 자세한 내용은 다음 웹 페이지를 확인하세요.

- 노드 버전 관리자 (nvm) - nvm repo on을 참조하십시오. GitHub

- 노드 패키지 관리자(npm) - npm 웹 사이트를 참조하세요.

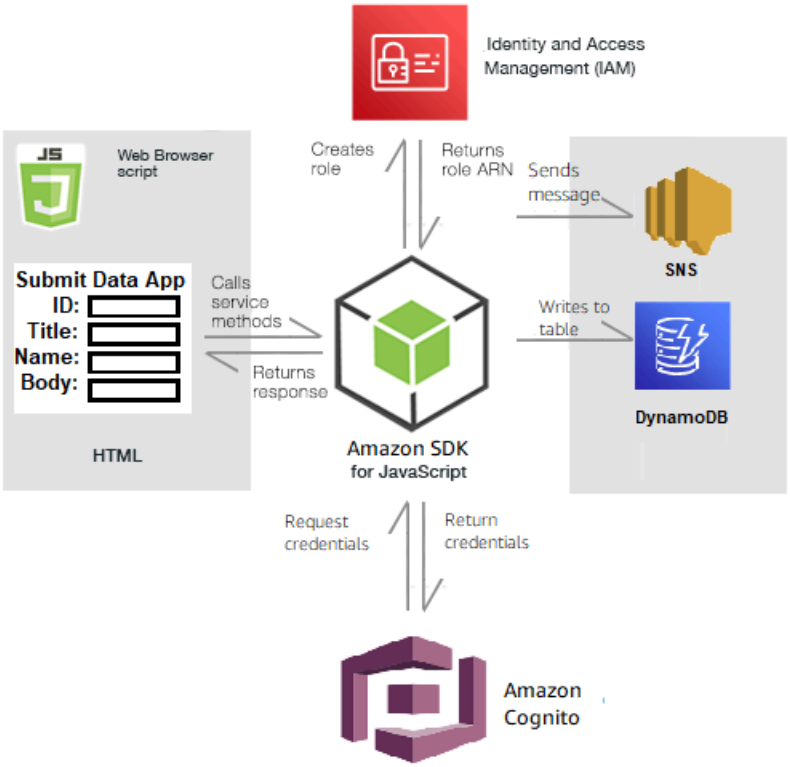
DynamoDB에 데이터를 제출하는 앱 빌드

이 교차 서비스 Node.js 자습서에서는 사용자가 Amazon DynamoDB 테이블에 데이터를 제출할 수 있게 지원하는 앱을 빌드하는 방법을 보여줍니다. 이 앱은 다음 서비스를 사용합니다.

- 인증 및 권한 부여를 위한 AWS Identity and Access Management(IAM) 및 Amazon Cognito
- 테이블 생성 및 업데이트를 위한 Amazon DynamoDB(DynamoDB)
- 사용자가 테이블을 업데이트하면 이를 앱 관리자에게 알리기 위한 Amazon Simple Notification Service(Amazon SNS)

시나리오

이 자습서에서 HTML 페이지는 Amazon DynamoDB 테이블에 데이터를 제출하기 위한 브라우저 기반 애플리케이션을 제공합니다. 앱은 Amazon SNS를 사용하여 사용자가 테이블을 업데이트할 때 이를 앱 관리자에게 알립니다.



앱을 빌드하려면 다음을 수행합니다.

1. 사전 조건
2. 리소스를 프로비저닝합니다.
3. HTML 생성
4. 브라우저 스크립트 생성
5. 다음 단계

사전 조건

다음 사전 조건 작업을 완료합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

AWS 리소스 생성

이 앱에는 다음 리소스가 필요합니다.

- 다음 권한이 있는 AWS Identity and Access Management(IAM) 미인증 Amazon Cognito 사용자 역할
  - sns:Publish
  - dynamodb:PutItem
- DynamoDB 테이블

이러한 리소스를 AWS 콘솔에서 수동으로 생성할 수 있지만, 이 자습서에 설명된 대로 AWS CloudFormation을 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

AWS CloudFormation을 사용하여 AWS 리소스 생성

AWS CloudFormation을 사용하여 AWS 인프라 배포를 예상한 대로 반복해서 생성하고 프로비저닝할 수 있습니다. AWS CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

AWS CLI를 사용하여 AWS CloudFormation 스택을 생성하려면 다음을 수행합니다.

1. AWS CLI 사용 설명서의 지침에 따라 AWS CLI를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 디렉터리에 이름이 `setup.yaml`인 파일을 생성하고 여기 GitHub의 내용을 해당 파일에 복사합니다.

> **Note**
>
> AWS CloudFormation 템플릿은 여기 GitHub에서 제공하는 AWS CDK를 사용하여 생성되었습니다. AWS CDK에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 가이드 섹션을 참조하세요.

3. 명령줄에서 다음 명령을 실행하여 *`STACK_NAME`*을 스택의 고유한 이름으로 바꾸고 *`REGION`*을 AWS 리전으로 바꿉니다.

> **Important**
>
> 스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

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

`create-stack` 명령 파라미터에 대한 자세한 내용은 AWS CLI 명령 참조 가이드 및 AWS CloudFormation 사용 설명서를 참조하세요.

생성된 리소스를 보려면 AWS 관리 콘솔에서 AWS CloudFormation을 열고 스택을 선택한 다음, 리소스 탭을 선택합니다.

4. 스택이 생성되면 테이블 채우기에 설명된 대로 AWS SDK for JavaScript를 사용하여 DynamoDB 테이블을 채웁니다.

테이블 채우기

테이블을 채우려면 먼저, 이름이 `libs`인 디렉터리를 생성하고 이 디렉터리 안에 이름이 `dynamoClient.js`인 파일을 생성한 다음, 아래 내용을 해당 파일에 붙여 넣습니다. *`REGION`*을 AWS 리전으로 바꾸고 *`IDENTITY_POOL_ID`*를 Amazon Cognito 자격 증명 풀 ID로 바꿉니다. 그러면 DynamoDB 클라이언트 객체가 생성됩니다.

```
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 DynaomDB service client object.
const dynamoClient = new DynamoDBClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { dynamoClient };
```

이 코드는 여기 GitHub에서 제공합니다.

다음으로, 프로젝트 폴더에 dynamoAppHelperFiles 폴더를 생성하고 그 안에 update-table.js 파일을 생성한 다음, 여기 GitHub의 내용을 해당 파일에 복사합니다.

```
// 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();
```

명령줄에서 다음 명령을 실행합니다.

```
node update-table.js
```

이 코드는 여기 GitHub에서 제공합니다.

앱의 프런트엔드 페이지 생성

여기서는 앱의 프런트엔드 HTML 브라우저 페이지를 만듭니다.

DynamoDBApp 디렉터리를 생성하고 이름이 `index.html`인 파일을 생성한 다음, 여기 GitHub에서 코드를 복사합니다. `script` 요소는 예에 필요한 모든 JavaScript가 포함된 `main.js` 파일을 추가합니다. 이 자습서에서는 나중에 `main.js` 파일을 생성합니다. `index.html`의 나머지 코드는 사용자가 입력하는 데이터를 캡처하는 브라우저 페이지를 만듭니다.

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

브라우저 스크립트 생성

먼저, 예에 필요한 서비스 클라이언트 객체를 생성합니다. `libs` 디렉터리를 생성하고 `snsClient.js`를 생성한 다음, 이 파일에 아래 코드를 붙여 넣습니다. 각각에서 *`REGION`* 및 *`IDENTITY_POOL_ID`*를 바꿉니다.

> **Note**
>
> AWS 리소스 생성 에서 생성한 Amazon Cognito 자격 증명 풀 ID를 사용하세요.

```
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 };
```

이 코드는 여기 GitHub에서 제공합니다.

이 예의 브라우저 스크립트를 생성하려면 DynamoDBApp이라는 폴더에 add_data.js 파일 이름으로 Node.js 모듈을 생성하고 아래 코드를 해당 모듈에 붙여 넣습니다. submitData 함수는 DynamoDB 테이블에 데이터를 제출하고 Amazon SNS를 사용하여 앱 관리자에게 SMS 텍스트를 보냅니다.

submitData 함수에서 대상 전화번호, 앱 인터페이스에 입력된 값, Amazon S3 버킷 이름에 대한 변수를 선언합니다. 다음으로, 테이블에 항목을 추가하기 위한 파라미터 객체를 생성합니다. 비어 있는 값이 없으면 submitData는 테이블에 항목을 추가하고 메시지를 보냅니다. window.submitData = submitData를 사용하여 브라우저에서 함수를 사용할 수 있도록 해야 합니다.

```
// 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;
```

이 코드 예는 여기 GitHub에서 찾을 수 있습니다.

마지막으로, 명령 프롬프트에서 다음을 실행하여 이 예의 JavaScript를 `main.js`라는 파일로 번들링합니다.

```
webpack add_data.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> Webpack 설치에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

앱을 실행하려면 브라우저에서 `index.html`을 엽니다.

리소스 삭제

이 자습서의 시작 부분에서 설명한 것처럼 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다. 다음과 같이 이 자습서의 AWS 리소스 생성 항목에서 생성한 AWS CloudFormation 스택을 삭제하면 됩니다.

1. AWS 관리 콘솔에서 AWS CloudFormation을 엽니다.
2. 스택 페이지를 열고 스택을 선택합니다.
3. Delete을 선택합니다.

더 많은 AWS 교차 서비스 예를 확인하려면 AWS SDK for JavaScript cross-service examples를 참조하세요.

인증된 사용자로 트랜스크립션 앱 빌드

이 자습서에서는 다음을 수행하는 방법을 알아봅니다.

- Amazon Cognito 자격 증명 풀을 사용하여 Amazon Cognito 사용자 풀과 페더레이션된 사용자를 허용하는 인증을 구현합니다.

- Amazon Transcribe를 사용하여 브라우저에서 음성 녹음을 텍스트로 기록하고 표시합니다.

시나리오

이 앱을 통해 사용자는 고유한 이메일과 사용자 이름으로 가입할 수 있습니다. 이메일을 확인하면 음성 메시지를 녹음할 수 있으며 이 음성 메시지는 자동으로 텍스트로 기록되어 앱에 표시됩니다.

작동 방식

이 앱은 두 개의 Amazon S3 버킷을 사용합니다. 하나는 애플리케이션 코드를 호스팅하고 다른 하나는 트랜스크립션을 저장합니다. 이 앱은 Amazon Cognito 사용자 풀을 사용하여 사용자를 인증합니다. 인증된 사용자는 필요한 AWS 서비스에 액세스할 수 있는 IAM 권한을 보유합니다.

사용자가 처음으로 음성 메시지를 녹음하면 Amazon S3는 트랜스크립션을 저장하기 위해 Amazon S3 버킷에 사용자 이름으로 고유한 폴더를 생성합니다. Amazon Transcribe는 음성 메시지를 텍스트로 변환하며 사용자의 폴더에 이 트랜스크립션을 JSON으로 저장합니다. 사용자가 앱을 새로 고치면 해당 트랜스크립션이 표시됩니다. 이 트랜스크립션은 다운로드하거나 삭제할 수 있습니다.

이 자습서를 완료하는 데 약 30분 정도 걸립니다.

단계

앱을 빌드하려면 다음을 수행합니다.

1. 사전 조건
2. AWS 리소스 생성
3. HTML 생성
4. 브라우저 스크립트 준비
5. 앱 실행
6. 리소스 삭제

사전 조건

- 이 노드 JavaScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

Important

이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

AWS 리소스 생성

이 섹션에서는 AWS Cloud Development Kit (AWS CDK)을 사용하여 이 앱의 AWS 리소스를 프로비저닝하는 방법을 설명합니다.

Note

AWS CDK은 클라우드 애플리케이션 리소스를 정의할 수 있게 지원하는 소프트웨어 개발 프레임워크입니다. 자세한 내용은 AWS Cloud Development Kit (AWS CDK)개발자 안내서를 참조하세요.

앱을 위한 리소스를 생성하려면 여기 GitHub의 템플릿을 사용하여 AWS Web Services Management Console 또는 AWS CLI를 통해 AWS CDK 스택을 생성합니다. 스택을 수정하는 방법 또는 자습서를 마쳤을 때 스택 및 관련 리소스를 삭제하는 방법에 대한 지침은 여기 GitHub를 참조하세요.

Note

스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

결과 스택은 다음 리소스를 자동으로 프로비저닝합니다.

- 인증된 사용자 역할이 있는 Amazon Cognito 자격 증명 풀
- Amazon S3 및 Amazon Transcribe에 대한 권한이 있는 IAM 정책은 인증된 사용자 역할에 연결됩니다.
- 사용자가 앱에 가입하고 로그인할 수 있게 지원하는 Amazon Cognito 사용자 풀
- 애플리케이션 파일을 호스팅하기 위한 Amazon S3 버킷

- 트랜스크립션을 저장하기 위한 Amazon S3 버킷

> **Important**
>
> 이 Amazon S3 버킷은 READ(LIST) 퍼블릭 액세스를 허용하므로 누구나 버킷 내의 객체를 나열하고 정보를 오용할 가능성이 있습니다. 자습서를 완료한 후 즉시 Amazon S3 버킷을 삭제하지 않는 경우 Amazon Simple Storage Service 사용 설명서의 Amazon S3의 보안 모범 사례를 준수하는 것이 좋습니다.

HTML 생성

`index.html` 파일을 생성하고 아래 내용을 복사하여 이 파일에 붙여 넣습니다. 이 페이지에는 음성 메시지 녹음을 위한 버튼 패널과 현재 사용자가 이전에 텍스트로 기록한 메시지를 표시하는 테이블이 있습니다. body 요소 끝부분에 있는 스크립트 태그는 앱의 모든 브라우저 스크립트가 포함된 `main.js`를 간접적으로 호출합니다. 이 자습서의 다음 단원에 설명된 대로 Webpack을 사용하여 main.js를 생성합니다.

```
<!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>
```

이 코드 예는 여기 GitHub에서 제공합니다.

브라우저 스크립트 준비

`index.html`, `recorder.js`, `helper.js`라는 세 가지 파일이 있으며, 이러한 파일은 Webpack을 사용하여 단일 `main.js`로 번들링하는 데 필요합니다. 이 섹션에서는 SDK for JavaScript를 사용하는 `index.js`의 함수만 자세히 설명하며, 이 코드는 GitHub의 여기에서 제공합니다.

> **Note**
>
> `recorder.js` 및 `helper.js`는 필요하기는 하지만, Node.js 코드를 포함하지 않으므로 GitHub의 여기 및 여기에서 제공하는 인라인 주석에 각각 설명되어 있습니다.

먼저, 파라미터를 정의합니다. `COGNITO_ID`는 이 자습서의 AWS 리소스 생성 주제에서 생성한 Amazon Cognito 사용자 풀의 엔드포인트입니다. 형식은 `cognito-idp.`*`AWS_REGION`*`.amazonaws.com/`*`USER_POOL_ID`*입니다. 사용자 풀 ID는 AWS 보안 인증 토큰의 *`ID_TOKEN`*이며, 이는 'helper.js' 파일의 `getToken` 함수에 의해 앱 URL에서 제거됩니다. 이 토큰은 `loginData` 변수에 전달되어 Amazon Transcribe 및 Amazon S3 클라이언트 객체에 로그인을 제공합니다. *`"REGION"`*을 AWS 리전으로 바꾸고, *`"BUCKET"`*을 으로 바꿉니다. *`"IDENTITY_POOL_ID"`*를 이 예에서 생성한 Amazon Cognito 자격 증명 풀의 샘플 페이지에 있는 `IdentityPoolId`로 바꿉니다. 이는 각 클라이언트 객체에도 전달됩니다.

```
// 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.
// 'COGINTO_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,
  }),
});
```

HTML 페이지가 로드되면 updateUserInterface는 사용자가 앱에 처음 로그인한 경우 Amazon S3 버킷에 사용자 이름으로 폴더를 생성합니다. 로그인한 것이 처음이 아니면 사용자의 이전 세션에 있는 기록으로 사용자 인터페이스를 업데이트합니다.

```
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 yieldUint8Chunks(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();
  }
}
```

사용자가 트랜스크립션을 위해 음성 메시지를 녹음하면 upload는 녹음 내용을 Amazon S3 버킷에 업로드합니다. 이 함수는 recorder.js 파일에서 직접적으로 호출됩니다.

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

deleteTranscription은 사용자 인터페이스에서 트랜스크립션을 삭제하고, deleteRow는 Amazon S3 버킷에서 기존 트랜스크립션을 삭제합니다. 이러한 메서드는 둘 다 사용자 인터페이스의 삭제 버튼에 의해 트리거됩니다.

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

마지막으로, 명령 프롬프트에서 다음을 실행하여 이 예의 JavaScript를 `main.js`라는 파일로 번들링합니다.

```
webpack index.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> webpack 설치에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

앱 실행

아래 위치에서 앱을 볼 수 있습니다.

```
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 Cognito를 사용하면 AWS Web Services Management Console에 링크를 제공하여 앱을 쉽게 실행할 수 있습니다. Amazon Cognito 사용자 풀의 앱 클라이언트 설정으로 이동하여 호스팅 UI 시작을 선택하기만 하면 됩니다. 앱의 URL 형식은 다음과 같습니다.

> **Important**
>
> 호스팅 UI의 기본 응답 유형은 '코드'입니다. 하지만 이 자습서는 '토큰' 응답 유형에 맞게 설계되었으므로 유형을 변경해야 합니다.

AWS 리소스 삭제

자습서를 마친 경우 불필요한 요금이 발생하지 않도록 리소스를 삭제해야 합니다. 두 Amazon S3 버킷에 모두 콘텐츠를 추가했으므로 수동으로 삭제해야 합니다. 그런 다음, AWS Web Services Management Console 또는 AWS CLI를 사용하여 나머지 리소스를 삭제할 수 있습니다. 스택을 수정하는 방법 또는 자습서를 마쳤을 때 스택 및 관련 리소스를 삭제하는 방법에 대한 지침은 여기 GitHub를 참조하세요.

API Gateway를 사용하여 Lambda 호출

REST, HTTP, WebSocket API를 대규모로 생성, 게시, 유지, 모니터링 및 보호하기 위한 AWS 서비스인 Amazon API Gateway를 사용하여 Lambda 함수를 간접적으로 호출할 수 있습니다. API 개발자는 AWS 또는 다른 웹 서비스를 비롯해 AWS 클라우드에 저장된 데이터에 액세스하는 API를 생성할 수 있습니다. API Gateway 개발자는 자체 클라이언트 애플리케이션에서 사용할 API를 생성할 수 있습니다. 자세한 내용은 Amazon API Gateway란 무엇입니까? 단원을 참조하세요.

AWS Lambda는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있게 하는 컴퓨팅 서비스입니다. 다양한 프로그래밍 언어로 Lambda 함수를 생성할 수 있습니다. AWS Lambda에 대한 자세한 정보는 AWS Lambda란 무엇입니까?를 참조하세요.

이 예제에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 특정 사용 사례를 수행하는 서로 다른 AWS 서비스를 호출합니다. 예를 들어 다음 그림과 같이 조직에서 1주년을 맞이하는 직원들에게 축하하는 모바일 문자 메시지를 보낸다고 가정해 보겠습니다.

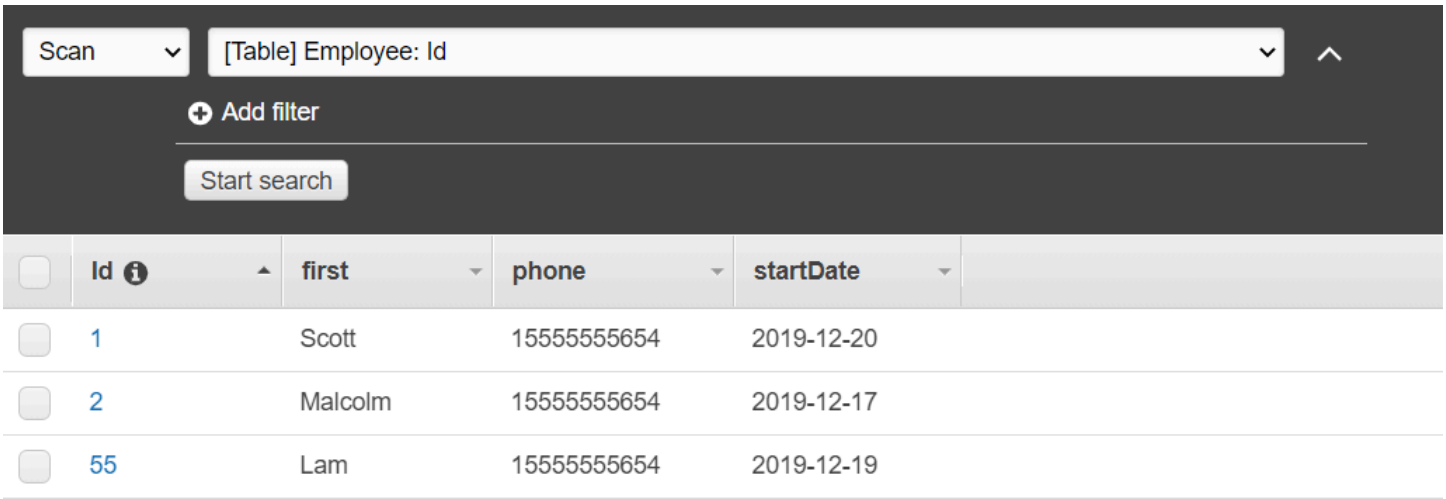
이 예를 완료하는 데 약 20분 정도 걸립니다.

이 예에서는 JavaScript 로직을 사용하여 이 사용 사례를 수행하는 솔루션을 생성하는 방법을 보여줍니다. 예를 들어 Lambda 함수를 사용하여 데이터베이스를 읽어 1주년 기념일을 맞이한 직원을 확인하는 방법, 데이터를 처리하고 문자 메시지를 보내는 방법을 모두 알아봅니다. 그런 다음, API Gateway를 사용하여 REST 엔드포인트를 통해 이 AWS Lambda 함수를 간접적으로 호출하는 방법을 알아봅니다. 예를 들어 다음 curl 명령을 사용하여 Lambda 함수를 간접적으로 호출할 수 있습니다.

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/
employee"
```

이 AWS 자습서에서는 다음 필드가 포함된 Employee라는 Amazon DynamoDB 테이블을 사용합니다.

- id - 테이블의 프라이머리 키입니다.
- firstName –직원의 이름입니다.
- phone –직원의 전화번호입니다.
- startDate – 직원의 시작 날짜입니다.



Important

완료 비용: 이 문서에 포함된 AWS 서비스는 AWS 프리 티어에 포함됩니다. 하지만 요금이 부과되지 않도록 하려면 이 예를 완료한 후에 모든 리소스를 종료해야 합니다.

앱을 빌드하려면 다음을 수행합니다.

1. 사전 조건 완료
2. AWS 리소스 생성
3. 브라우저 스크립트 준비
4. Lambda 함수 생성 및 업로드
5. Lambda 함수 배포

6. 앱 실행
7. 리소스 삭제

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

AWS 리소스 생성

이 자습서를 시작하려면 다음 리소스가 필요합니다.

- 이전 그림에 나와 있는 필드와 `Id`라는 키가 있는 `Employee`라는 Amazon DynamoDB 테이블. 이 사용 사례를 테스트하려는 유효한 휴대폰을 포함해 올바른 데이터를 입력했는지 확인하세요. 자세한 내용은 테이블 생성을 참조하세요.
- Lambda 함수를 실행하기 위한 권한이 연결된 IAM 역할.
- Lambda 함수를 호스팅하는 Amazon S3 버킷.

이러한 리소스를 수동으로 생성할 수 있지만, 이 자습서에 설명된 대로 AWS CloudFormation을 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

AWS CloudFormation을 사용하여 AWS 리소스 생성

AWS CloudFormation을 사용하여 AWS 인프라 배포를 예상한 대로 반복해서 생성하고 프로비저닝할 수 있습니다. AWS CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

AWS CLI를 사용하여 AWS CloudFormation 스택을 생성하려면 다음을 수행합니다.

1. AWS CLI 사용 설명서의 지침에 따라 AWS CLI를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 디렉터리에 이름이 `setup.yaml`인 파일을 생성하고 여기 GitHub의 내용을 해당 파일에 복사합니다.

> **Note**
>
> AWS CloudFormation 템플릿은 여기 GitHub에서 제공하는 AWS CDK를 사용하여 생성되었습니다. AWS CDK에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 가이드 섹션을 참조하세요.

3. 명령줄에서 다음 명령을 실행하여 *STACK_NAME*을 스택의 고유한 이름으로 바꿉니다.

> **Important**
>
> 스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

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

`create-stack` 명령 파라미터에 대한 자세한 내용은 AWS CLI 명령 참조 가이드 및 AWS CloudFormation 사용 설명서를 참조하세요.

4. 다음으로, 테이블 채우기 절차에 따라 테이블을 채웁니다.

테이블 채우기

테이블을 채우려면 먼저, 이름이 `libs`인 디렉터리를 생성하고 이 디렉터리 안에 이름이 `dynamoClient.js`인 파일을 생성한 다음, 아래 내용을 해당 파일에 붙여 넣습니다.

```
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 };
```

이 코드는 여기 GitHub에서 제공합니다.

다음으로, 프로젝트 폴더의 루트 디렉터리에 이름이 `populate-table.js`인 파일을 생성하고 여기 GitHub의 내용을 해당 파일에 복사합니다. 항목 중 하나에 대해 phone 속성의 값을 E.164 형식의 유효한 휴대폰 번호로 바꾸고 `startDate`의 값을 오늘 날짜로 바꿉니다.

명령줄에서 다음 명령을 실행합니다.

```
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();
```

이 코드는 여기 GitHub에서 제공합니다.

AWS Lambda 함수 생성

SDK 구성

`libs` 디렉터리에서 이름이 `snsClient.js` 및 `lambdaClient.js`인 파일을 생성하고 아래 내용을 각각 해당 파일에 붙여 넣습니다.

```
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 };
```

*REGION*을 AWS 리전으로 바꿉니다. 이 코드는 여기 GitHub에서 제공합니다.

```
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 };
```

*REGION*을 AWS 리전으로 바꿉니다. 이 코드는 여기 GitHub에서 제공합니다.

먼저, 필수 AWS SDK for JavaScript(v3) 모듈과 명령을 가져옵니다. 그런 다음, 오늘 날짜를 계산하여 파라미터에 할당합니다. 셋째, ScanCommand에 대한 파라미터를 생성합니다. *TABLE_NAME*을 이 예시의 AWS 리소스 생성 섹션에서 생성한 테이블 이름으로 바꿉니다.

다음 코드 조각은 이 단계를 보여줍니다. (전체 예제는 Lambda 함수 번들링 섹션을 참조하세요.)

```
"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",
};
```

DynamoDB 테이블 스캔

먼저, Amazon SNS PublishCommand를 사용하여 텍스트 메시지를 게시하는 sendText라는 async/await 함수를 생성합니다. 그런 다음, DynamoDB 테이블을 스캔하여 오늘이 근무 기념일인 직원을 찾

은 후 sendText 함수를 직접적으로 호출하여 해당 직원에게 문자 메시지를 보내는 try 블록 패턴을 추가합니다. 오류가 발생하면 catch 블록이 직접적으로 호출됩니다.

다음 코드 조각은 이 단계를 보여줍니다. (전체 예제는 Lambda 함수 번들링 섹션을 참조하세요.)

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

Lambda 함수 번들링

이 항목에서는 이 예에 필요한 AWS SDK for JavaScript 모듈과 mylambdafunction.ts 모듈을 index.js라는 번들 파일로 묶는 방법을 설명합니다.

1. webpack을 아직 설치하지 않았다면 이 예의 사전 필수 작업에 따라 설치합니다.

> **Note**
>
> Webpack에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

2. 명령줄에서 다음을 실행하여 이 예시의 JavaScript를 <index.js>라는 파일로 번들링합니다.

```
webpack mylambdafunction.ts --mode development --target node --devtool false --
output-library-target umd -o index.js
```

> **Important**
>
> 출력 이름이 index.js인 것에 주목하세요. 이는 Lambda 함수가 작동하려면 index.js 핸들러가 있어야 하기 때문입니다.

3. 번들 출력 파일, index.js를 mylambdafunction.zip이라는 ZIP 파일로 압축합니다.
4. 이 자습서의 AWS 리소스 생성 항목에서 생성한 Amazon S3 버킷에 mylambdafunction.zip을 업로드합니다.

Lambda 함수 배포

프로젝트의 루트에서 lambda-function-setup.ts 파일을 생성하고 아래 내용을 해당 파일에 붙여 넣습니다.

*BUCKET_NAME*을 Lambda 함수의 ZIP 버전을 업로드한 Amazon S3 버킷 이름으로 바꿉니다. *ZIP_FILE_NAME*을 Lambda 함수의 ZIP 버전 이름으로 바꿉니다. *ROLE*을 이 자습서의 AWS 리소스 생성 항목에서 생성한 IAM 역할의 Amazon 리소스 번호(ARN)로 바꿉니다. *LAMBDA_FUNCTION_NAME*을 Lambda 함수 이름으로 바꿉니다.

```
// 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();
```

명령줄에 다음을 입력하여 Lambda 함수를 배포합니다.

```
node lambda-function-setup.ts
```

이 코드 예는 여기 GitHub에서 제공합니다.

Lambda 함수를 간접적으로 호출하도록 API Gateway 구성

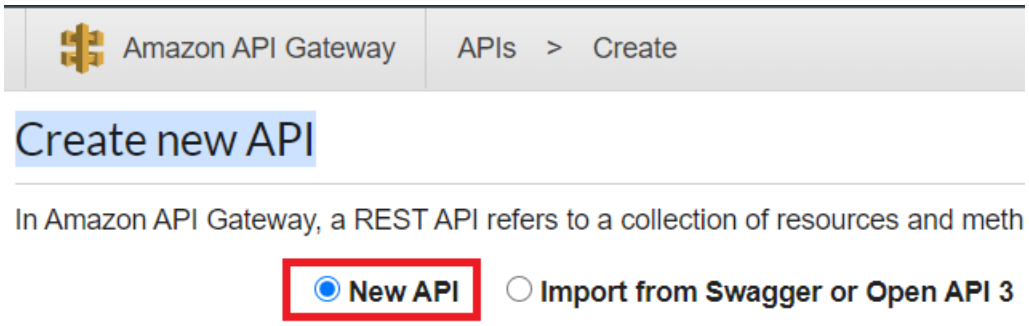
앱을 빌드하려면 다음을 수행합니다.

1. REST API 생성
2. API Gateway 메서드 테스트
3. API Gateway 메서드 배포

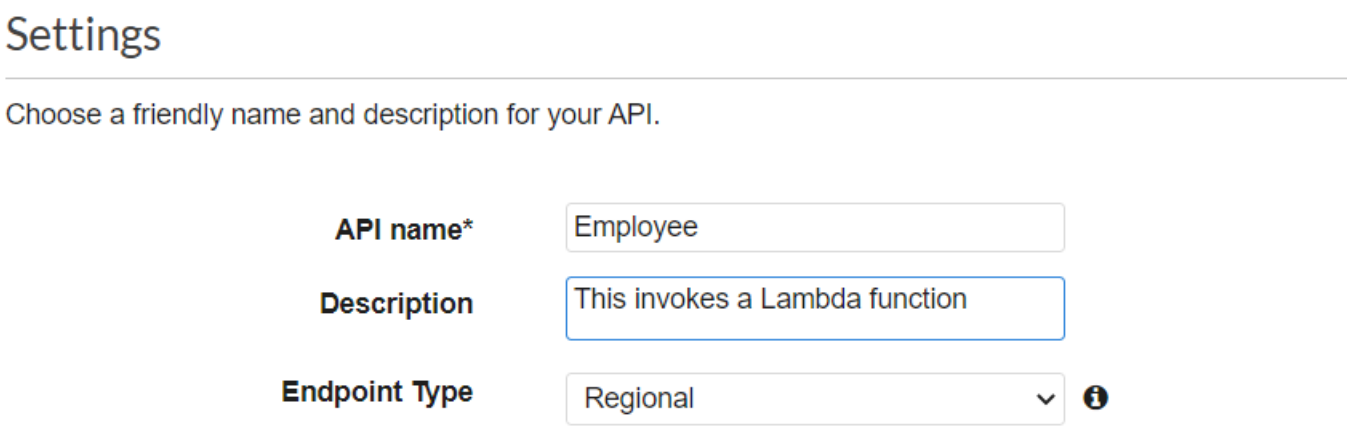
REST API 생성

API Gateway 콘솔을 사용하여 Lambda 함수의 REST 엔드포인트를 생성할 수 있습니다. 완료하면 RESTful 호출을 사용하여 Lambda 함수를 간접적으로 호출할 수 있습니다.

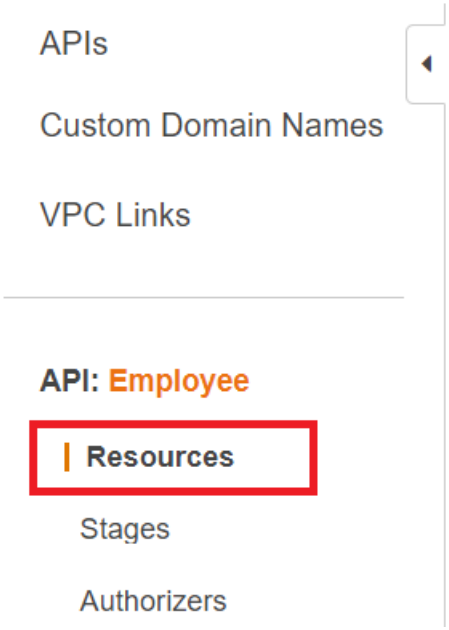
1. Amazon API Gateway 콘솔에 로그인합니다.
2. REST API에서 빌드를 선택합니다.
3. 새 API를 선택합니다.



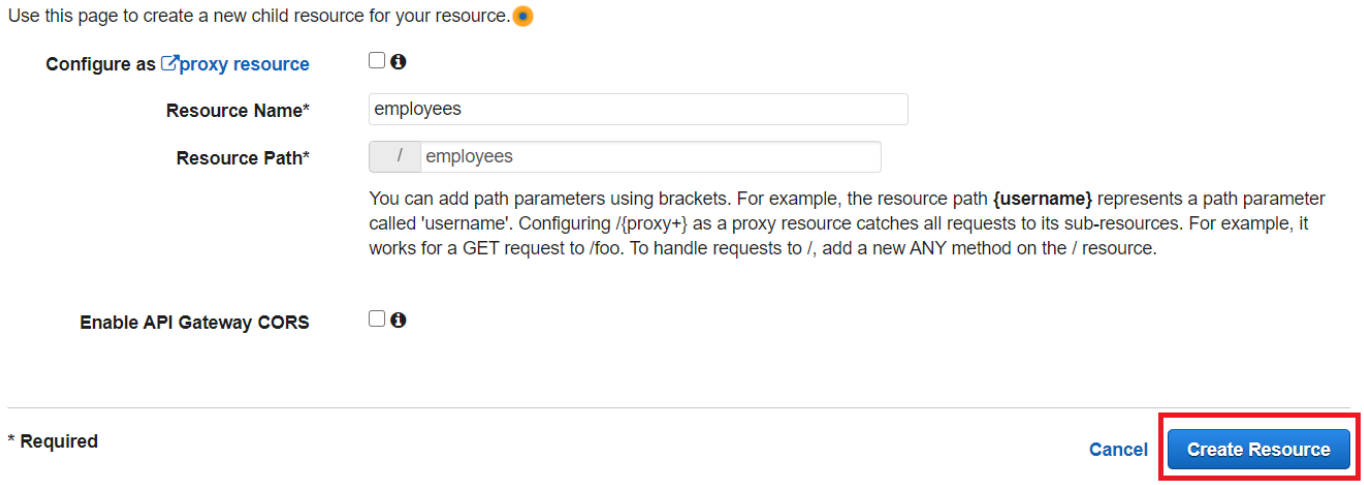
4. 직원을 API 이름으로 지정하고 설명을 입력합니다.



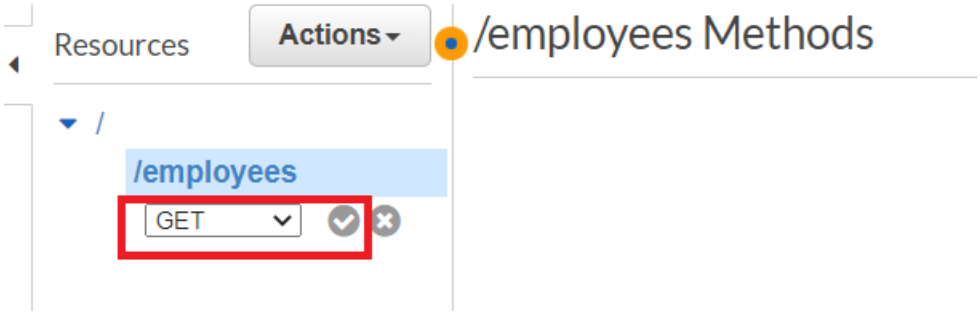
5. API 생성(Create API)을 선택합니다.
6. 직원 섹션에서 리소스를 선택합니다.



7. 이름 필드에서 직원을 지정합니다.
8. 리소스 생성을 선택합니다.
9. 작업 드롭다운에서 리소스 생성을 선택합니다.



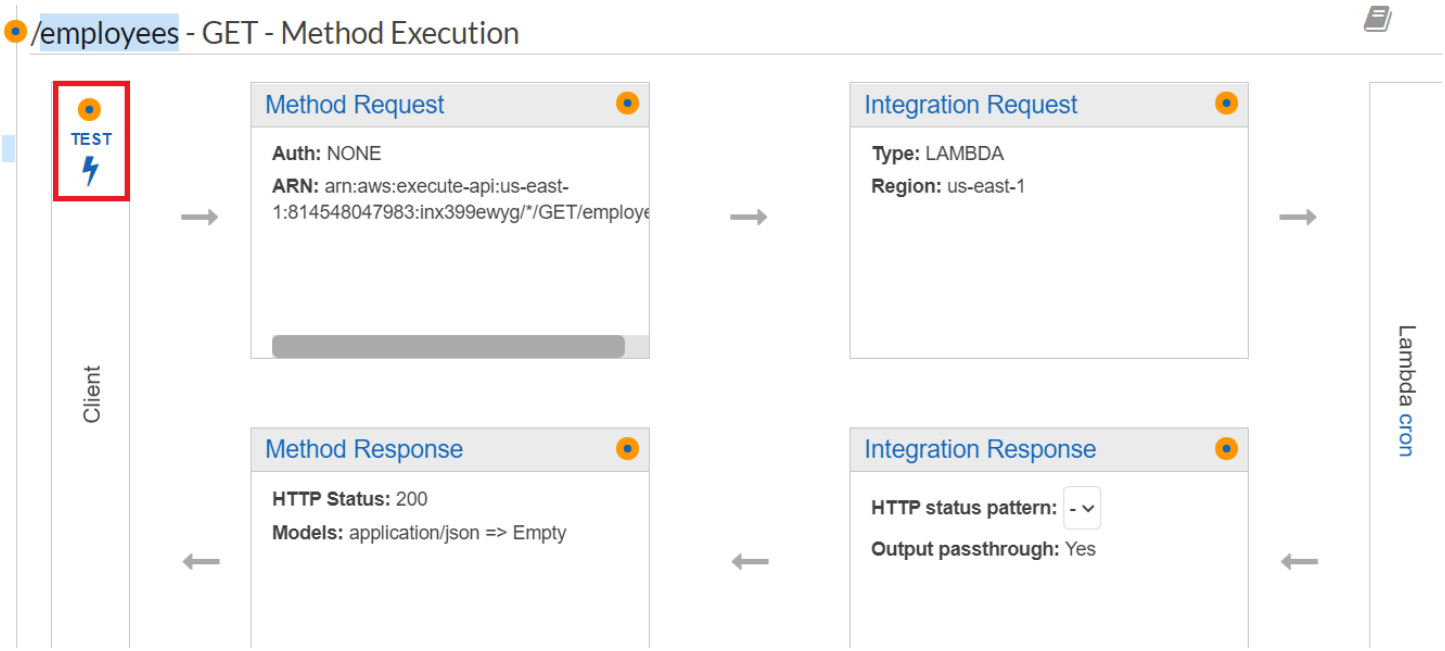
10. /employees를 선택하고, 작업에서 메서드 생성을 선택한 다음, /employees 아래의 드롭다운 메뉴에서 GET을 선택합니다. 체크 표시 아이콘을 선택합니다.



11. Lambda 함수를 선택하고 Lambda 함수 이름으로 mylambdafunction을 입력합니다. 저장을 선택합니다.

API Gateway 메서드 테스트

자습서의 이 시점에서 mylambdafunction Lambda 함수를 간접적으로 호출하는 API Gateway 메서드를 테스트할 수 있습니다. 메서드를 테스트하려면 다음 그림과 같이 테스트를 선택합니다.

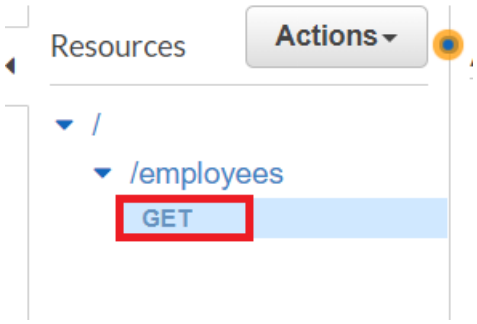


Lambda 함수가 간접적으로 호출되면 로그 파일을 보고 성공 메시지를 확인할 수 있습니다.

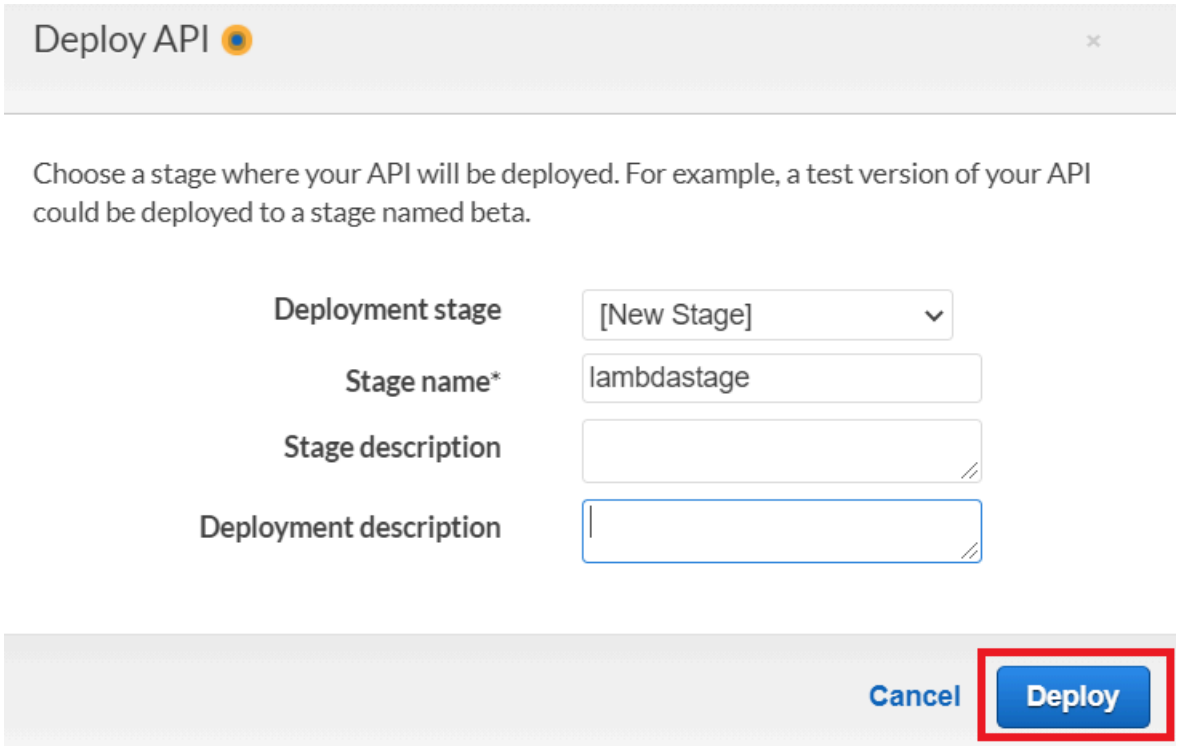
API Gateway 메서드 배포

테스트가 성공하면 Amazon API Gateway 콘솔에서 메서드를 배포할 수 있습니다.

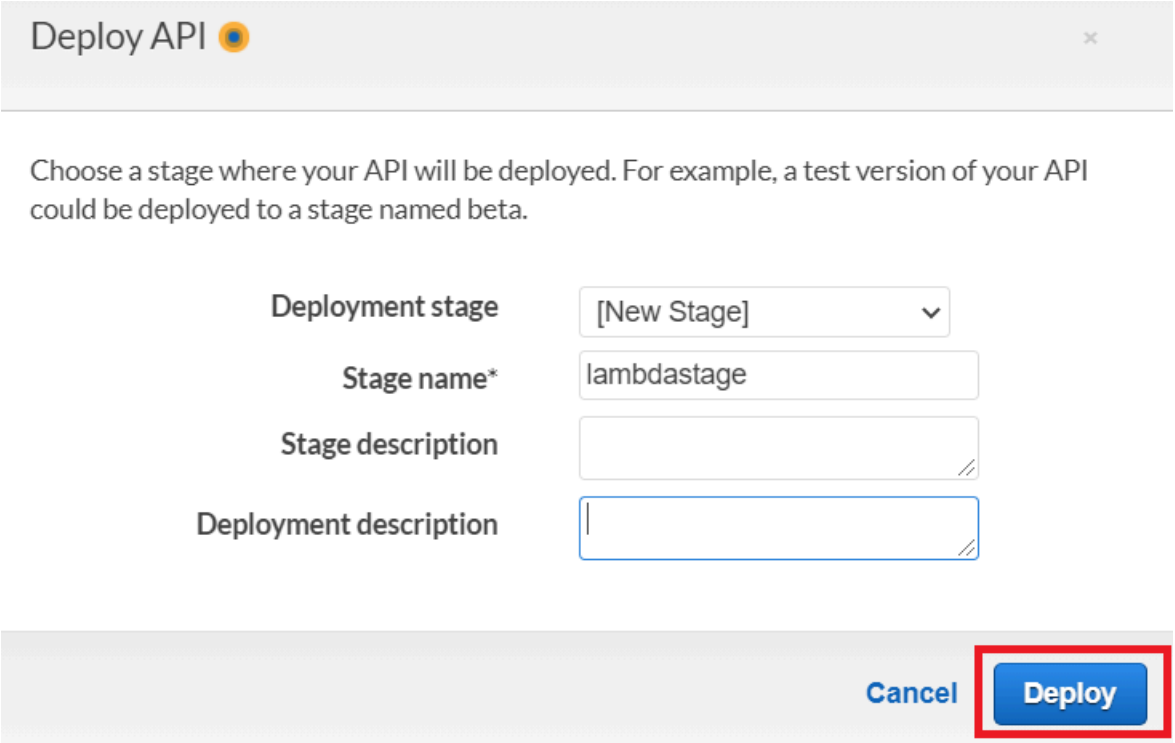
1. GET을 선택합니다.



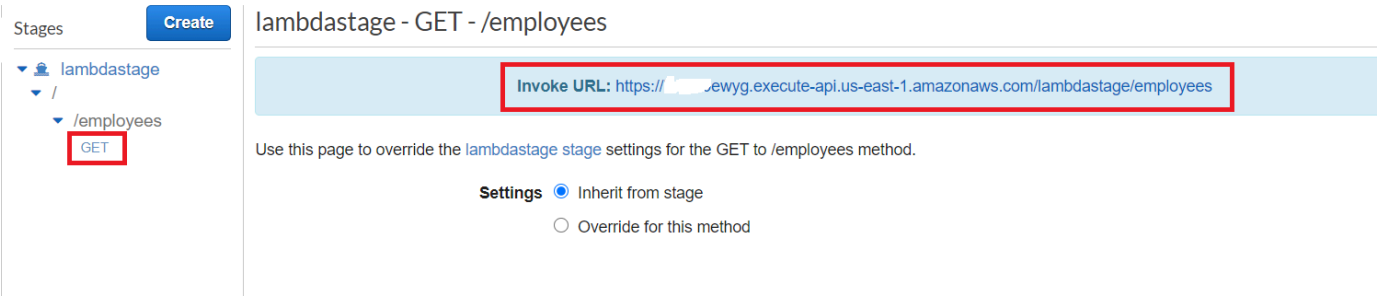
2. 작업 드롭다운에서 API 배포를 선택합니다.



3. API 배포 양식을 작성하고 배포를 선택합니다.



4. 변경 사항 저장을 선택합니다.
5. GET을 다시 선택하면 URL이 변경된 것을 확인할 수 있습니다. 이는 Lambda 함수를 간접적으로 호출하는 데 사용할 수 있는 호출 URL입니다.



리소스 삭제

축하합니다! AWS SDK for JavaScript를 사용하여 Amazon API Gateway를 통해 Lambda 함수를 간접적으로 호출했습니다. 이 자습서의 시작 부분에서 설명한 것처럼 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다. 다음과 같이 이 자습서의 AWS 리소스 생성 주제에서 생성한 AWS CloudFormation 스택을 삭제하면 됩니다.

1. AWS 관리 콘솔에서 AWS CloudFormation을 엽니다.
2. 스택 페이지를 열고 스택을 선택합니다.

3. 삭제를 선택합니다.

AWS SDK for JavaScript를 사용하여 AWS 서버리스 워크플로 생성

Step Functions, AWS SDK for Java, AWS Step Functions를 사용하여 AWS 서버리스 워크플로를 생성할 수 있습니다. 각 워크플로 단계는 AWS Lambda 함수를 사용하여 구현됩니다. Lambda는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있게 하는 컴퓨팅 서비스입니다. Step Functions는 Lambda 함수와 기타 AWS 서비스를 결합할 수 있는 서버리스 오케스트레이션 서비스로, 비즈니스 크리티컬 애플리케이션을 구축합니다.

> **Note**
>
> 다양한 프로그래밍 언어로 Lambda 함수를 생성할 수 있습니다. 이 자습서에서는 Lambda Java API를 사용하여 Lambda 함수를 구현했습니다. Lambda에 관한 자세한 내용은 AWS Lambda란 무엇인가요? 단원을 참조하세요.

이 자습서에서는 조직에 대한 지원 티켓을 생성하는 워크플로를 만듭니다. 각 워크플로 단계는 티켓에 대한 작업을 수행합니다. 이 자습서에서는 JavaScript를 사용하여 워크플로 데이터를 처리하는 방법을 보여줍니다. 예를 들어 워크플로에 전달된 데이터를 읽는 방법, 단계 간에 데이터를 전달하는 방법, 워크플로에서 AWS 서비스를 호출하는 방법을 알아봅니다.

완료 비용: 이 문서에 포함된 AWS 서비스는 AWS 프리 티어에 포함됩니다.

참고: 더 이상 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다.

주제

- 사전 필수 작업
- AWS 리소스 생성
- 워크플로 생성
- Lambda 함수 생성
- 워크플로에 Lambda 함수 추가
- Step Functions 콘솔을 사용하여 워크플로 실행
- AWS 리소스 삭제

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. GitHub의 지침을 따릅니다.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

AWS 리소스 생성

이 자습서를 시작하려면 다음 리소스가 필요합니다.

- Id라는 키가 있는 Case라는 Amazon DynamoDB 테이블.
- Lambda 함수를 호출하는 데 사용되는 `lambda-support`라는 IAM 역할. 이 역할에는 Lambda 함수에서 Amazon DynamoDB 및 Amazon Simple Email Service 서비스를 호출할 수 있게 지원하는 정책이 있습니다.
- 워크플로를 호출하는 데 사용되는 `workflow-support`라는 IAM 역할.
- Lambda 함수를 호스팅하는 Amazon S3 버킷.

이러한 리소스를 수동으로 생성할 수 있지만, 이 자습서에 설명된 대로 AWS Cloud Development Kit (AWS CDK)(AWS CDK)를 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

AWS CloudFormation을 사용하여 AWS 리소스 생성

AWS CloudFormation을 사용하여 AWS 인프라 배포를 예상한 대로 반복해서 생성하고 프로비저닝할 수 있습니다. AWS CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

AWS CloudFormation 스택을 생성하려면 다음을 수행합니다.

1. AWS CLI 사용 설명서의 지침에 따라 AWS CLI를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 디렉터리에 이름이 `setup.yaml`인 파일을 생성하고 여기 GitHub의 내용을 해당 파일에 복사합니다.

> **Note**
>
> AWS CloudFormation 템플릿은 여기 GitHub에서 제공하는 AWS CDK를 사용하여 생성되었습니다. AWS CDK에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 가이드 섹션을 참조하세요.

3. 명령줄에서 다음 명령을 실행하여 *STACK_NAME*을 스택의 고유한 이름으로 바꿉니다.

> **Important**
>
> 스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

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

`create-stack` 명령 파라미터에 대한 자세한 내용은 AWS CLI 명령 참조 가이드 및 AWS CloudFormation 사용 설명서를 참조하세요.

Amazon Web Services Management Console을 사용하여 AWS 리소스를 생성합니다.

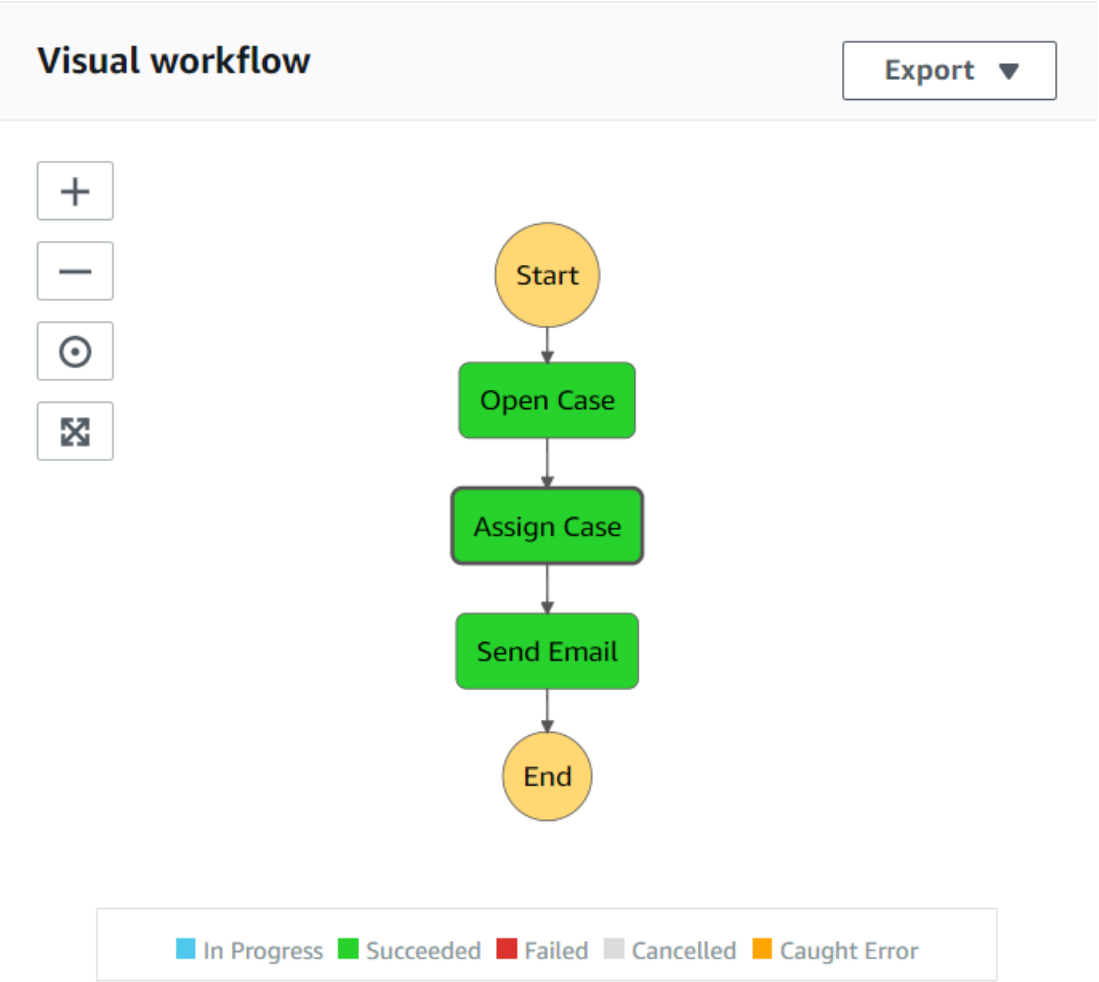
콘솔에서 앱 리소스를 생성하려면 AWS CloudFormation 사용 설명서의 지침을 따릅니다. 제공된 템플릿을 사용하여 이름이 `setup.yaml`인 파일을 생성하고 여기 GitHub의 내용을 복사합니다.

> **Important**
>
> 스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

AWS CloudFormation 대시보드에서 스택을 열고 리소스 탭을 선택하여 콘솔에서 리소스 목록을 확인합니다. 자습서에는 이러한 정보가 필요합니다.

워크플로 생성

다음 그림은 이 자습서를 통해 생성할 워크플로를 보여줍니다.



워크플로의 각 단계에서 발생하는 상황은 다음과 같습니다.

+ 시작 – 워크플로를 시작합니다.

+ 사례 열기 - 지원 티켓 ID 값을 워크플로에 전달하여 처리합니다.

+ 사례 할당 - 지원 사례를 직원에게 할당하고 데이터를 DynamoDB 테이블에 저장합니다.

+ 이메일 전송 - Amazon Simple Email Service(Amazon SES)를 사용하여 직원에게 새 티켓이 있음을 알리는 이메일 메시지를 보냅니다.

+ 종료 – 워크플로를 중지합니다.

Step Functions를 사용하여 서버리스 워크플로 생성

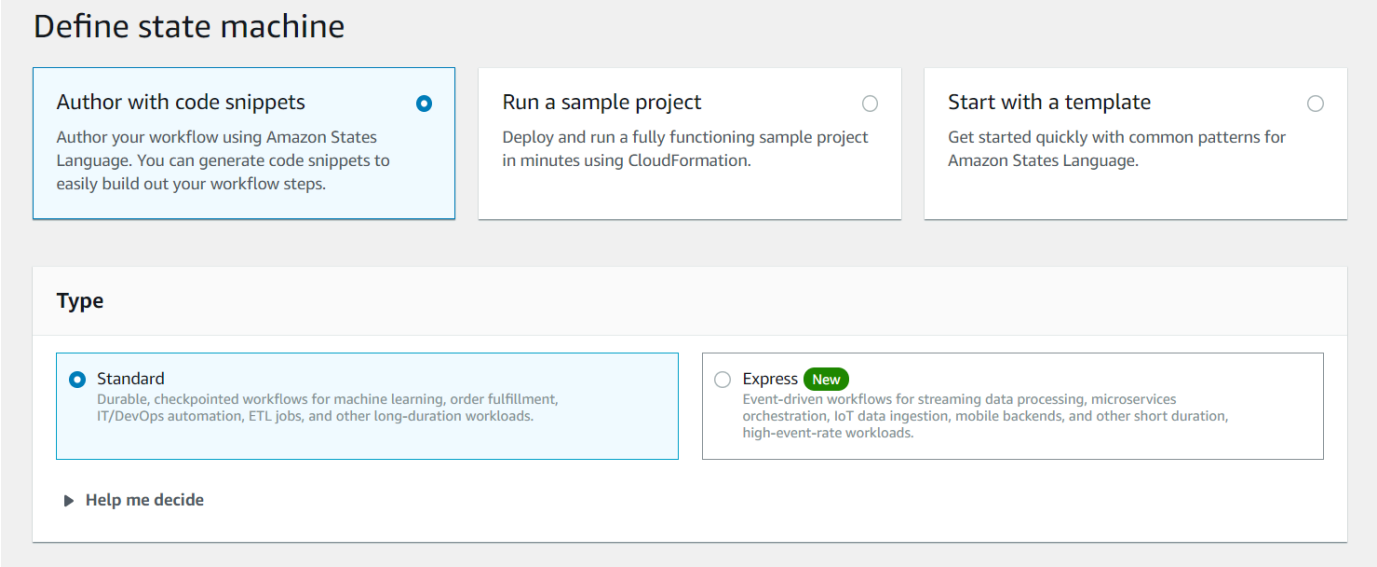
지원 티켓을 처리하는 워크플로를 생성할 수 있습니다. Step Functions를 사용하여 워크플로를 정의하려면 Amazon States Language(JSON 기반) 문서를 생성하여 상태 머신을 정의합니다. Amazon

States Language 문서는 각 단계를 설명합니다. 문서를 정의하면 Step Functions가 워크플로를 시각적으로 나타냅니다. 다음 그림은 Amazon States Language 문서와 워크플로의 시각적 표현을 보여줍니다.

워크플로는 단계 간에 데이터를 전달할 수 있습니다. 예를 들어 사례 열기 단계에서는 (워크플로에 전달된) 사례 ID 값을 처리하고 해당 값을 사례 할당 단계에 전달합니다. 이 자습서의 뒷부분에서는 Lambda 함수에서 데이터 값을 읽고 처리하는 애플리케이션 로직을 생성합니다.

워크플로 생성 방법

1. Amazon Web Services 콘솔을 엽니다.
2. 상태 머신 생성을 선택합니다.
3. [코드 조각으로 작성]을 선택합니다. 유형 영역에서 표준을 선택합니다.



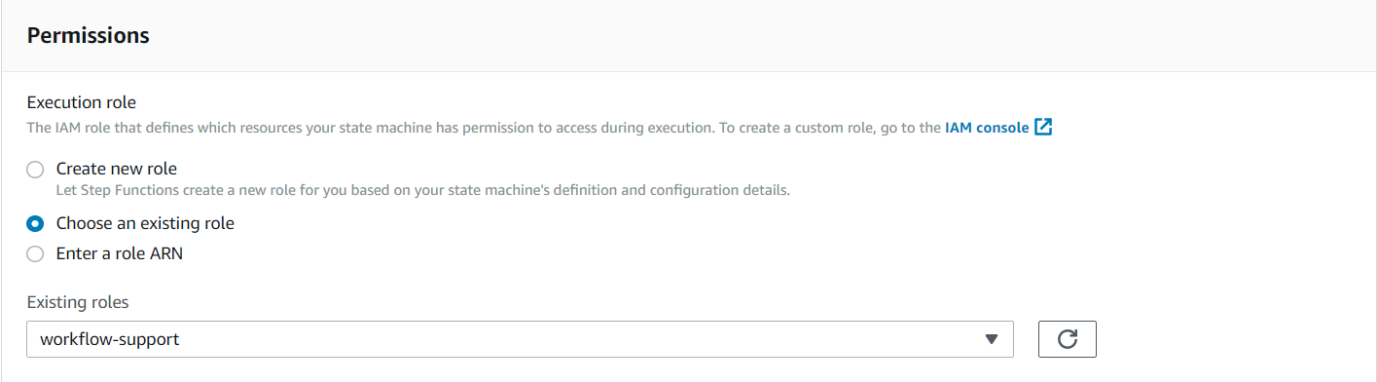
4. 다음 코드를 입력하여 Amazon States Language 문서를 지정합니다.

```
{
"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",
```

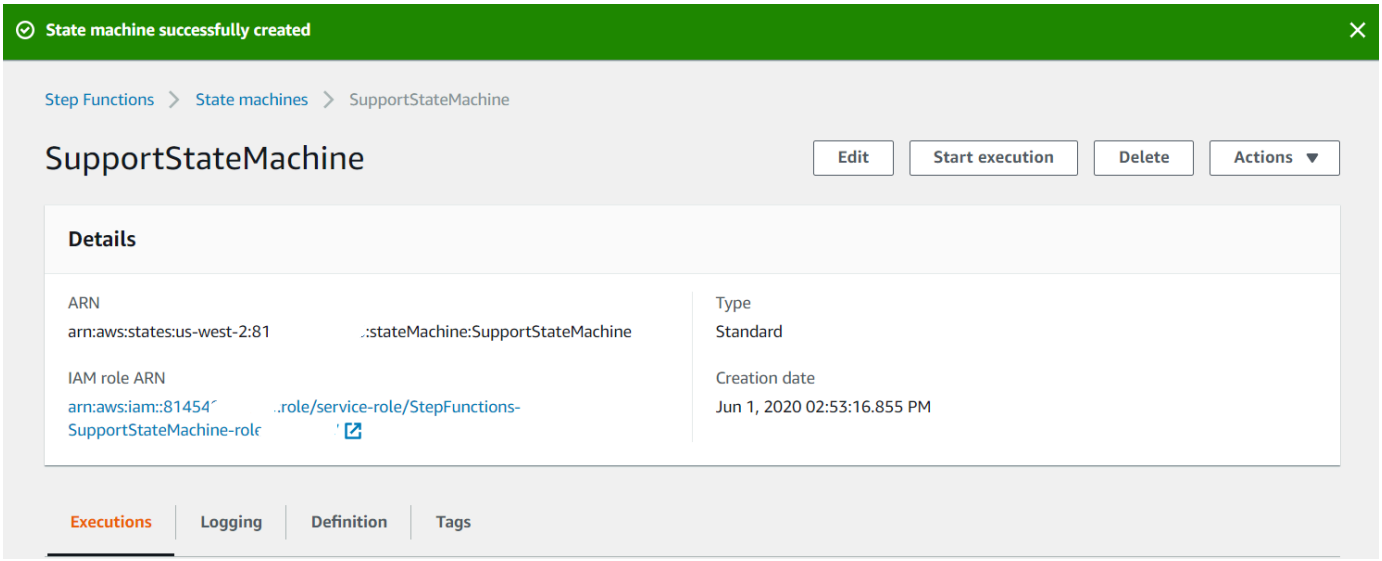
```
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"Next": "Send Email"
},
"Send Email": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"End": true
}
}
}
```

> **Note**
>
> Lambda 리소스 값과 관련된 오류는 걱정하지 마세요. 이 자습서의 뒷부분에서 이러한 값을 업데이트합니다.

5. 다음을 선택합니다.
6. 이름 필드에 SupportStateMachine을 입력합니다.
7. 권한 섹션에서 기존 역할 선택을 선택합니다.
8. workflow-support(생성한 IAM 역할)를 선택합니다.



9. 상태 머신 생성을 선택합니다. 상태 머신이 성공적으로 생성되었음을 알리는 메시지가 나타납니다.



Lambda 함수 생성

Lambda 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예에는 각 Lambda 함수에 해당하는 세 가지 워크플로 단계가 있습니다.

다음 단원에 설명된 대로 이러한 Lambda 함수를 생성합니다.

- getId Lambda 함수 – 티켓 ID 값을 처리하는 워크플로의 첫 번째 단계로 사용됩니다.
- addItem Lambda 클래스 - 티켓을 직원에게 할당하고 DynamoDB 데이터베이스에 데이터를 저장하는 워크플로의 두 번째 단계로 사용됩니다.
- sendemail Lambda 클래스 - Amazon SES를 사용하여 직원에게 티켓에 관해 알리는 이메일 메시지를 보내는 워크플로의 세 번째 단계로 사용됩니다.

getId Lambda 함수

워크플로의 두 번째 단계에 전달된 티켓 ID 값을 반환하는 Lambda 함수를 생성합니다.

```
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);
   }
};
```

명령줄에 다음을 입력하여 webpack을 사용해 파일을 index.js라는 파일로 번들링합니다.

```
webpack getid.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

그런 다음 ZIP 파일 이름 getid.js.zip으로 index.js를 압축합니다. 이 예시의 주제에서 생성한 Amazon S3 버킷에 ZIP 파일을 업로드합니다.

이 코드 예는 여기 GitHub에서 제공합니다.

addItem Lambda 클래스

티켓을 할당할 직원을 선택한 다음, Case라는 DynamoDB 테이블에 티켓 데이터를 저장하는 Lambda 함수를 생성합니다.

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

명령줄에 다음을 입력하여 webpack을 사용해 파일을 `index.js`라는 파일로 번들링합니다.

```
webpack additem.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

그런 다음 ZIP 파일 이름 `additem.js.zip`으로 `index.js`를 압축합니다. 이 예시의 주제에서 생성한 Amazon S3 버킷에 ZIP 파일을 업로드합니다.

이 코드 예는 여기 GitHub에서 제공합니다.

sendemail Lambda 클래스

새 티켓에 관해 알리는 이메일을 보내는 Lambda 함수를 생성합니다. 두 번째 단계에서 전달받은 이메일 주소가 사용됩니다.

```
// 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 recepient = 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 = `<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: [recepient],
    },
    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);
  }
};
```

명령줄에 다음을 입력하여 webpack을 사용해 파일을 `index.js`라는 파일로 번들링합니다.

```
webpack sendemail.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

그런 다음 ZIP 파일 이름 `sendemail.js.zip`으로 `index.js`를 압축합니다. 이 예시의 주제에서 생성한 Amazon S3 버킷에 ZIP 파일을 업로드합니다.

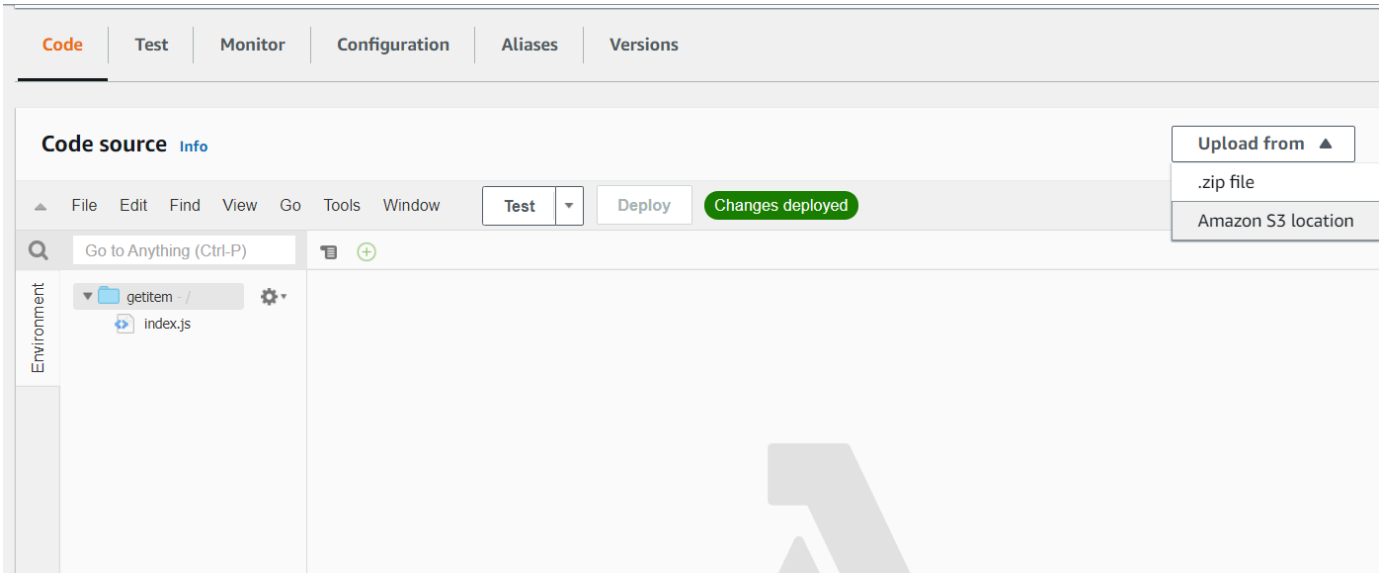
이 코드 예는 여기 GitHub에서 제공합니다.

Lambda 함수 배포

`getid` Lambda 함수를 배포하려면 다음을 수행합니다.

1. Amazon Web Services 콘솔에서 Lambda 콘솔을 엽니다.
2. 함수 생성을 선택합니다.
3. 새로 작성을 선택합니다.
4. 기본 정보 섹션에서 이름으로 getid를 입력합니다.
5. 런타임에서 Node.js 14x를 선택합니다.
6. 기존 역할 사용을 선택한 다음, lambda-support(생성한 IAM 역할)를 선택합니다.
7. 함수 생성(Create function)을 선택합니다.

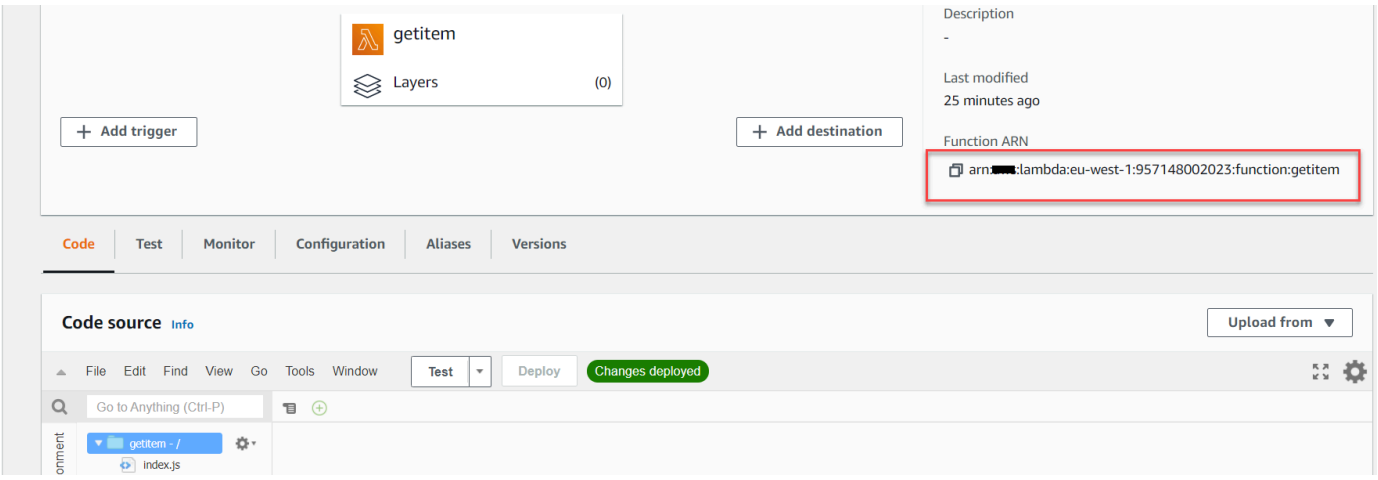
8. 에서 업로드 – Amazon S3 위치를 선택합니다.
9. 업로드를 선택하고 에서 업로드 – Amazon S3 위치를 선택한 다음, Amazon S3 링크 URL을 입력합니다.



10. Save를 선택합니다.
11. 새 Lambda 함수를 위해 additem.js.zip 및 sendemail.js.zip에 대해 이 절차를 반복합니다. 완료하면 Amazon States Language 문서에서 참조할 수 있는 세 가지 Lambda 함수가 있습니다.

워크플로에 Lambda 함수 추가

1. Lambda 콘솔을 엽니다. 오른쪽 상단에서 Lambda Amazon 리소스 이름(ARN) 값을 볼 수 있습니다.



2. 값을 복사한 다음, Step Functions 콘솔에 있는 Amazon States Language 문서의 1단계에 붙여 넣습니다.

3. 사례 할당 및 이메일 전송 단계의 리소스를 업데이트합니다. 이는 AWS SDK for Java를 사용하여 생성한 Lambda 함수를 Step Functions를 사용하여 생성한 워크플로에 연결하는 방법입니다.

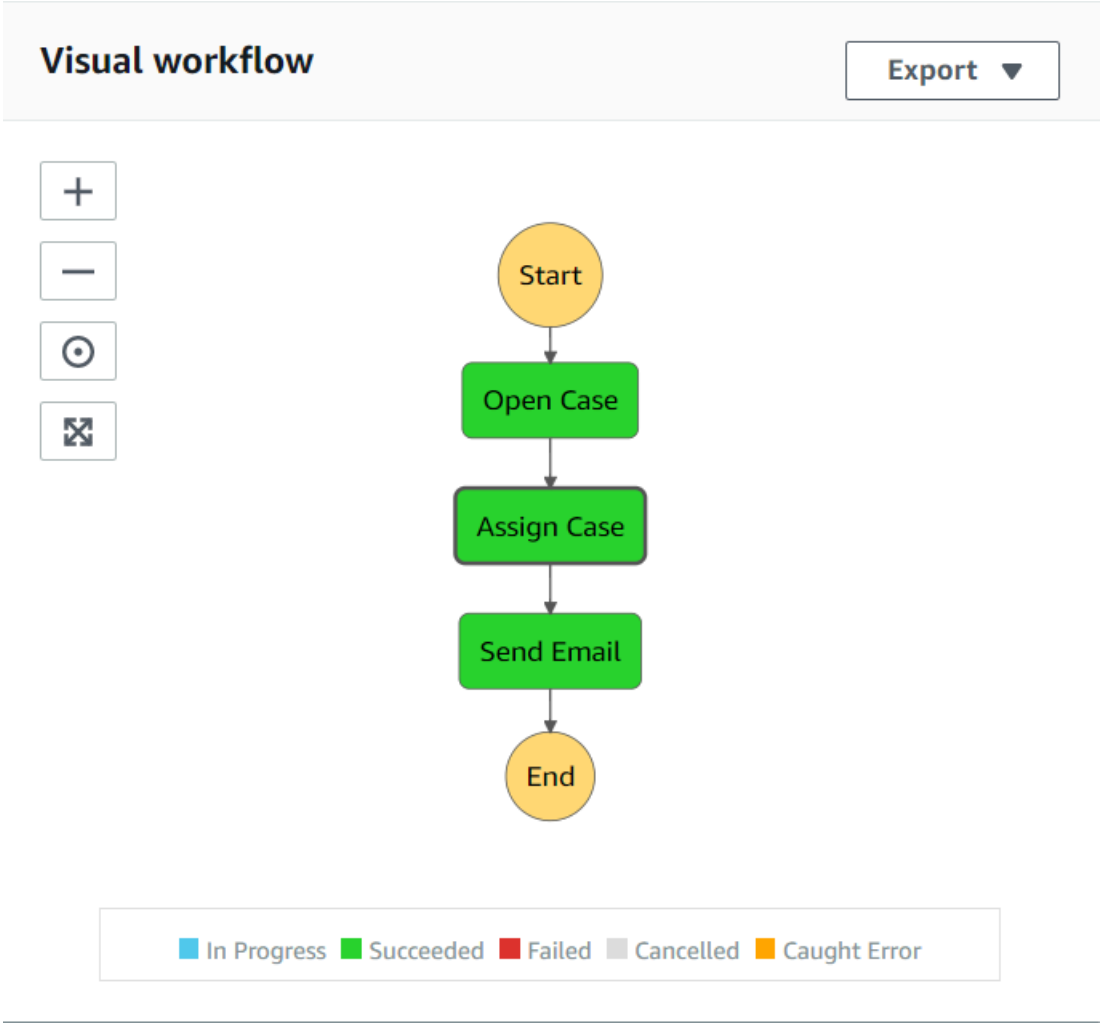
Step Functions 콘솔을 사용하여 워크플로 실행

Step Functions 콘솔에서 워크플로를 호출할 수 있습니다. 실행은 JSON 입력을 받습니다. 이 예에서는 다음 JSON 데이터를 워크플로에 전달할 수 있습니다.

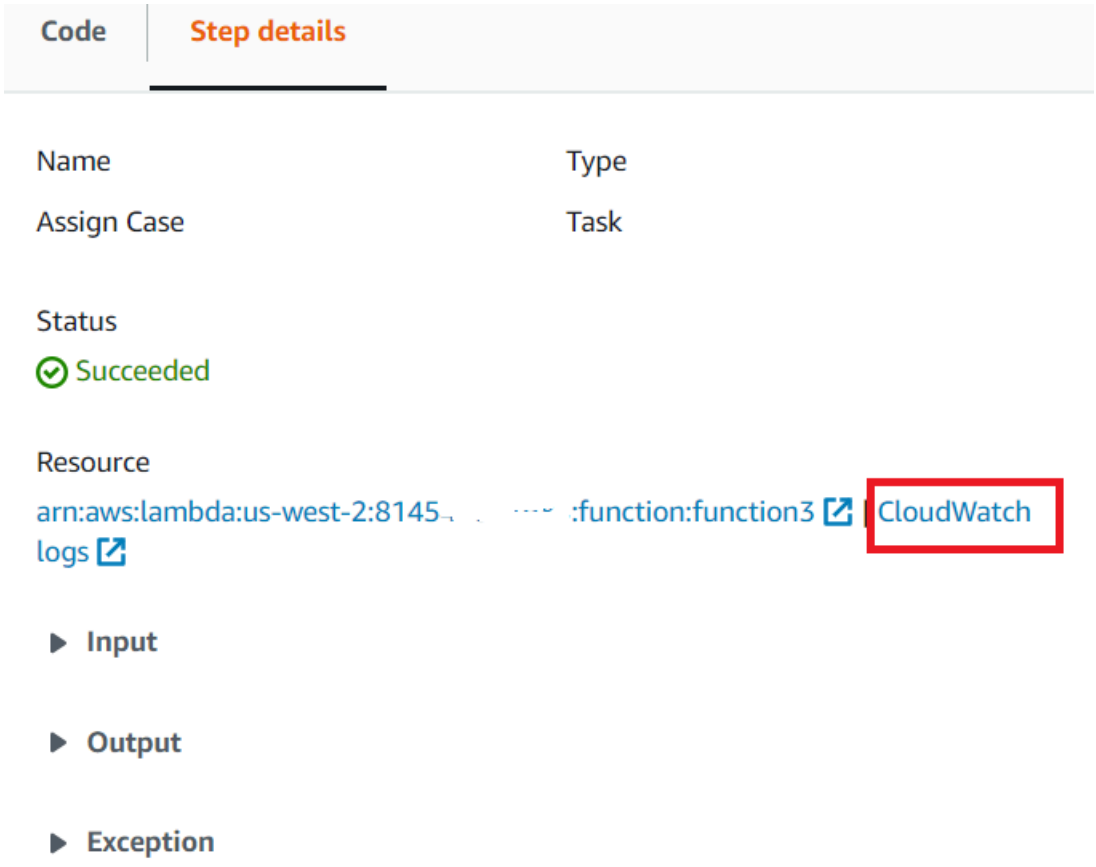
```
{
"inputCaseID": "001"
}
```

워크플로를 실행하려면 다음을 수행합니다.

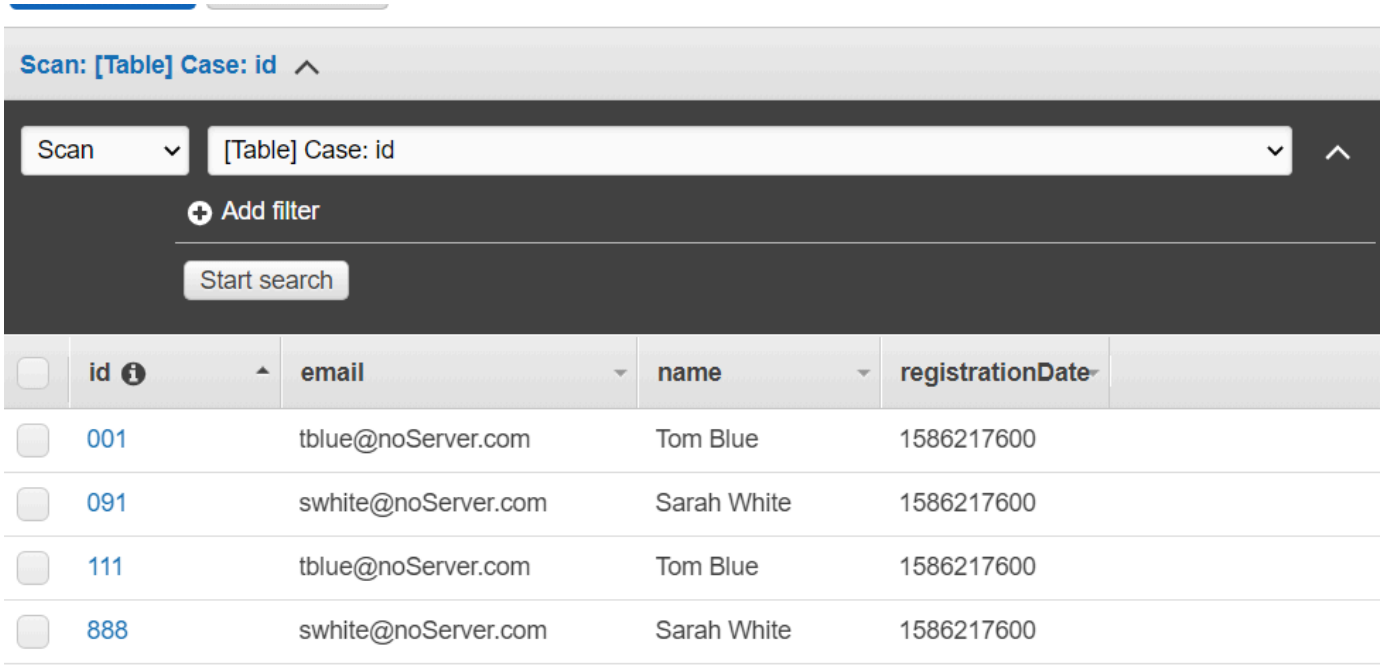
1. Step Functions 콘솔에서 실행 시작을 선택합니다.
2. 입력 섹션에서 JSON 데이터를 전달합니다. 워크플로를 확인합니다. 각 단계가 완료되면 녹색으로 바뀝니다.



3. 단계가 빨간색으로 바뀌면 오류가 발생한 것입니다. 단계를 클릭하면 오른쪽에서 액세스할 수 있는 로그를 볼 수 있습니다.



워크플로가 완료되면 DynamoDB 테이블의 데이터를 볼 수 있습니다.



AWS 리소스 삭제

축하합니다. AWS SDK for Java를 사용하여 AWS 서버리스 워크플로를 생성했습니다. 이 자습서의 시작 부분에서 설명한 것처럼 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다. 다음과 같이 이 자습서의 AWS 리소스 생성 주제에서 생성한 AWS CloudFormation 스택을 삭제하면 됩니다.

1. AWS 관리 콘솔에서 AWS CloudFormation을 엽니다.
2. 스택 페이지를 열고 스택을 선택합니다.
3. 삭제를 선택합니다.

AWS Lambda 함수를 실행하도록 예약된 이벤트 생성

Amazon Event를 사용하여 AWS Lambda 함수를 호출하는 스케줄링된 CloudWatch 이벤트를 생성할 수 있습니다. cron 표현식을 사용하여 Lambda 함수가 호출되는 시기를 스케줄링하도록 CloudWatch 이벤트를 구성할 수 있습니다. 예를 들어 평일마다 Lambda 함수를 호출하도록 CloudWatch 이벤트를 예약할 수 있습니다.

AWS Lambda는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있게 하는 컴퓨팅 서비스입니다. 다양한 프로그래밍 언어로 Lambda 함수를 생성할 수 있습니다. AWS Lambda에 대한 자세한 정보는 AWS Lambda란 무엇입니까?를 참조하세요.

이 자습서에서는 Lambda 런타임 API를 사용하여 Lambda 함수를 생성합니다. JavaScript 이 예제에서는 특정 사용 사례를 수행하는 서로 다른 AWS 서비스를 호출합니다. 예를 들어 다음 그림과 같이 조직에서 1주년을 맞이하는 직원들에게 축하하는 모바일 문자 메시지를 보낸다고 가정해 보겠습니다.

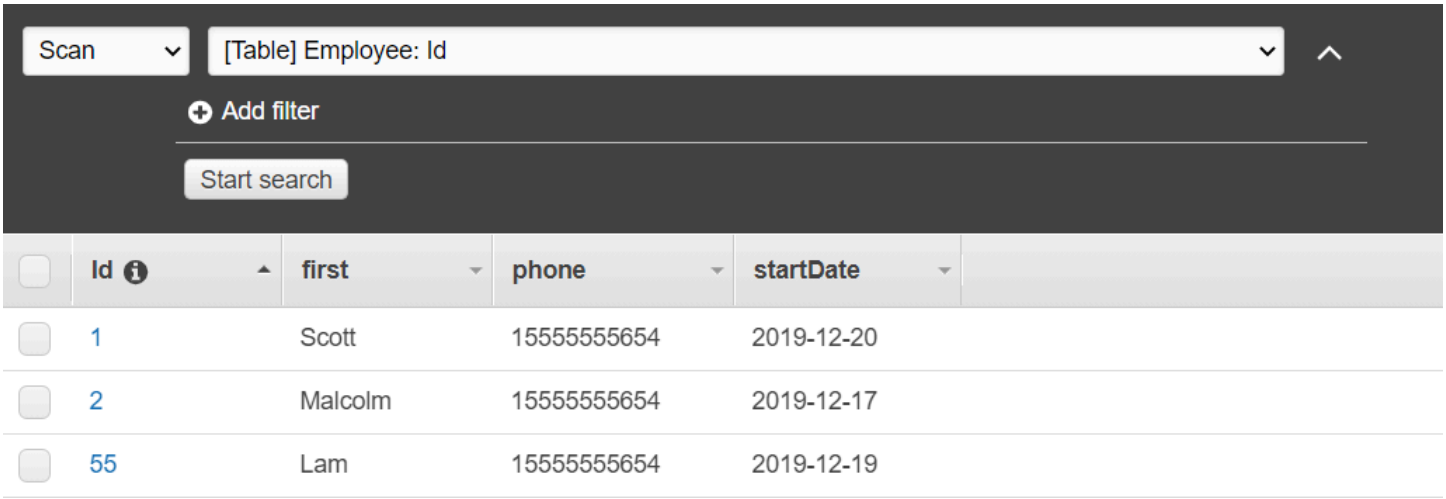
이 자습서를 완료하는 데 약 20분 정도 걸립니다.

이 자습서에서는 JavaScript 로직을 사용하여 이 사용 사례를 수행하는 솔루션을 생성하는 방법을 보여줍니다. 예를 들어 Lambda 함수를 사용하여 데이터베이스를 읽어 1주년 기념일을 맞이한 직원을 확인

하는 방법, 데이터를 처리하고 문자 메시지를 보내는 방법을 모두 알아봅니다. 그런 다음, cron 표현식을 사용하여 평일마다 Lambda 함수를 간접적으로 호출하는 방법을 알아봅니다.

이 AWS 자습서에서는 다음 필드가 포함된 Employee라는 Amazon DynamoDB 테이블을 사용합니다.

- id - 테이블의 프라이머리 키입니다.
- firstName –직원의 이름입니다.
- phone –직원의 전화번호입니다.
- startDate – 직원의 시작 날짜입니다.



> **Important**
>
> 완료 비용: 이 문서에 포함된 AWS 서비스는 AWS 프리 티어에 포함됩니다. 하지만 요금이 부과되지 않도록 하려면 이 자습서를 완료한 후에 모든 리소스를 종료해야 합니다.

앱을 빌드하려면 다음을 수행합니다.

1. 사전 조건 완료
2. AWS 리소스 생성
3. 브라우저 스크립트 준비
4. Lambda 함수 생성 및 업로드
5. Lambda 함수 배포
6. 앱 실행

7. 리소스 삭제

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 Node.js TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. 의 지침을 따르십시오 GitHub.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

AWS 리소스 생성

이 자습서를 시작하려면 다음 리소스가 필요합니다.

- 이전 그림에 나와 있는 필드와 Id라는 키가 있는 Employee라는 Amazon DynamoDB 테이블. 이 사용 사례를 테스트하려는 유효한 휴대폰을 포함해 올바른 데이터를 입력했는지 확인하세요. 자세한 내용은 테이블 생성을 참조하세요.
- Lambda 함수를 실행하기 위한 권한이 연결된 IAM 역할.
- Lambda 함수를 호스팅하는 Amazon S3 버킷.

이러한 리소스를 수동으로 생성할 수 있지만, 이 자습서에 설명된 대로 AWS CloudFormation을 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

AWS CloudFormation을 사용하여 AWS 리소스 생성

AWS CloudFormation을 사용하여 AWS 인프라 배포를 예상한 대로 반복해서 생성하고 프로비저닝할 수 있습니다. AWS CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

AWS CLI를 사용하여 AWS CloudFormation 스택을 생성하려면 다음을 수행합니다.

1. AWS CLI 사용 설명서의 지침에 따라 AWS CLI를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 `setup.yaml` 디렉터리에 이름이 지정된 파일을 만들고 여기에 콘텐츠를 복사합니다. GitHub

> **Note**
>
> AWS CloudFormation템플릿은 여기에서 AWS CDK 사용할 수 있는 것을 사용하여 생성되었습니다 GitHub. AWS CDK에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 가이드 섹션을 참조하세요.

3. 명령줄에서 다음 명령을 실행하여 *STACK_NAME*을 스택의 고유한 이름으로 바꿉니다.

> **Important**
>
> 스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

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

`create-stack` 명령 파라미터에 대한 자세한 내용은 AWS CLI 명령 참조 가이드 및 AWS CloudFormation 사용 설명서를 참조하세요.

AWS CloudFormation 대시보드에서 스택을 열고 리소스 탭을 선택하여 콘솔에서 리소스 목록을 확인합니다. 자습서에는 이러한 정보가 필요합니다.

4. 스택이 생성되면 DynamoDB 테이블 채우기에 설명된 대로 AWS SDK for JavaScript를 사용하여 DynamoDB 테이블을 채웁니다.

DynamoDB 테이블 채우기

테이블을 채우려면 먼저, 이름이 `libs`인 디렉터리를 생성하고 이 디렉터리 안에 이름이 `dynamoClient.js`인 파일을 생성한 다음, 아래 내용을 해당 파일에 붙여 넣습니다.

```
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 };
```

이 코드는 여기에서 확인할 수 있습니다 GitHub.

그런 다음, 프로젝트 폴더의 루트 `populate-table.js` 디렉터리에 이름이 지정된 파일을 만들고 여기에 내용을 복사합니다. GitHub 항목 중 하나에 대해 phone 속성의 값을 E.164 형식의 유효한 휴대폰 번호로 바꾸고 `startDate`의 값을 오늘 날짜로 바꿉니다.

명령줄에서 다음 명령을 실행합니다.

```
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();
```

이 코드는 여기에서 확인할 수 있습니다 GitHub.

AWS Lambda 함수 생성

SDK 구성

먼저, 필수 AWS SDK for JavaScript(v3) 모듈과 명령(`DynamoDBClient`, DynamoDB `ScanCommand`, `SNSClient`, Amazon SNS `PublishCommand` 명령)을 가져옵니다. *`REGION`*을 AWS 리전으로 바꿉니다. 그런 다음, 오늘 날짜를 계산하여 파라미터에 할당합니다. 다음으로, `ScanCommand`.Replace *`TABLE_NAME`*의 파라미터를 이 예의 AWS 리소스 생성 섹션에서 생성한 테이블 이름을 사용해 생성합니다.

다음 코드 조각은 이 단계를 보여줍니다. (전체 예제는 Lambda 함수 번들링 섹션을 참조하세요.)

```
"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",
};
```

DynamoDB 테이블 스캔

먼저, Amazon SNS `PublishCommand`를 사용하여 텍스트 메시지를 게시하는 `sendText`라는 async/await 함수를 생성합니다. 그런 다음, DynamoDB 테이블을 스캔하여 오늘이 근무 기념일인 직원을 찾은 후 `sendText` 함수를 직접적으로 호출하여 해당 직원에게 문자 메시지를 보내는 `try` 블록 패턴을 추가합니다. 오류가 발생하면 `catch` 블록이 직접적으로 호출됩니다.

다음 코드 조각은 이 단계를 보여줍니다. (전체 예제는 Lambda 함수 번들링 섹션을 참조하세요.)

```
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);
  }
};
```

Lambda 함수 번들링

이 항목에서는 이 예에 필요한 AWS SDK for JavaScript 모듈과 `mylambdafunction.js` 모듈을 `index.js`라는 번들 파일로 묶는 방법을 설명합니다.

1. webpack을 아직 설치하지 않았다면 이 예의 사전 필수 작업에 따라 설치합니다.

   > **Note**
   >
   > Webpack에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

2. 명령줄에서 다음을 실행하여 이 예제의 내용을 JavaScript 라는 파일로 번들로 묶습니다<index.js>.

   ```
   webpack mylamdbafunction.js --mode development --target node --devtool false --
   output-library-target umd -o index.js
   ```

   > **Important**
   >
   > 출력 이름이 `index.js`인 것에 주목하세요. 이는 Lambda 함수가 작동하려면 `index.js` 핸들러가 있어야 하기 때문입니다.

3. 번들 출력 파일, `index.js`를 `my-lambda-function.zip`이라는 ZIP 파일로 압축합니다.
4. 이 자습서의 AWS 리소스 생성 항목에서 생성한 Amazon S3 버킷에 `mylambdafunction.zip`을 업로드합니다.

mylambdafunction.js에 대한 전체 브라우저 스크립트 코드는 다음과 같습니다.

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

Lambda 함수 배포

프로젝트의 루트에서 `lambda-function-setup.js` 파일을 생성하고 아래 내용을 해당 파일에 붙여 넣습니다.

*BUCKET_NAME*을 Lambda 함수의 ZIP 버전을 업로드한 Amazon S3 버킷 이름으로 바꿉니다. *ZIP_FILE_NAME*을 Lambda 함수의 ZIP 버전 이름으로 바꿉니다. *IAM_ROLE_ARN*을 이 자습서의 AWS 리소스 생성 항목에서 생성한 IAM 역할의 Amazon 리소스 번호(ARN)로 바꿉니다. *LAMBDA_FUNCTION_NAME*을 Lambda 함수 이름으로 바꿉니다.

```
// 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();
```

명령줄에 다음을 입력하여 Lambda 함수를 배포합니다.

```
node lambda-function-setup.js
```

이 코드 예제는 여기에서 확인할 수 GitHub 있습니다.

Lambda 함수를 CloudWatch 호출하도록 구성합니다.

Lambda 함수를 CloudWatch 호출하도록 구성하려면:

1. Lambda 콘솔에서 함수 페이지를 엽니다.
2. Lambda 함수를 선택합니다.
3. Designer에서 트리거 추가를 선택합니다.
4. 트리거 유형을 Events/로 설정합니다. CloudWatch EventBridge
5. 규칙에서 새 규칙 생성을 선택합니다.

6. 규칙 이름과 규칙 설명을 입력합니다.
7. 규칙 유형에서 예약 표현식을 선택합니다.
8. 예약 표현식 필드에 cron 표현식을 입력합니다. 예를 들어 cron(0 12 ? * MON-FRI *)를 입력합니다.
9. 추가를 선택합니다.

> **Note**
>
> 자세한 내용은 이벤트와 함께 Lambda CloudWatch 사용을 참조하십시오.

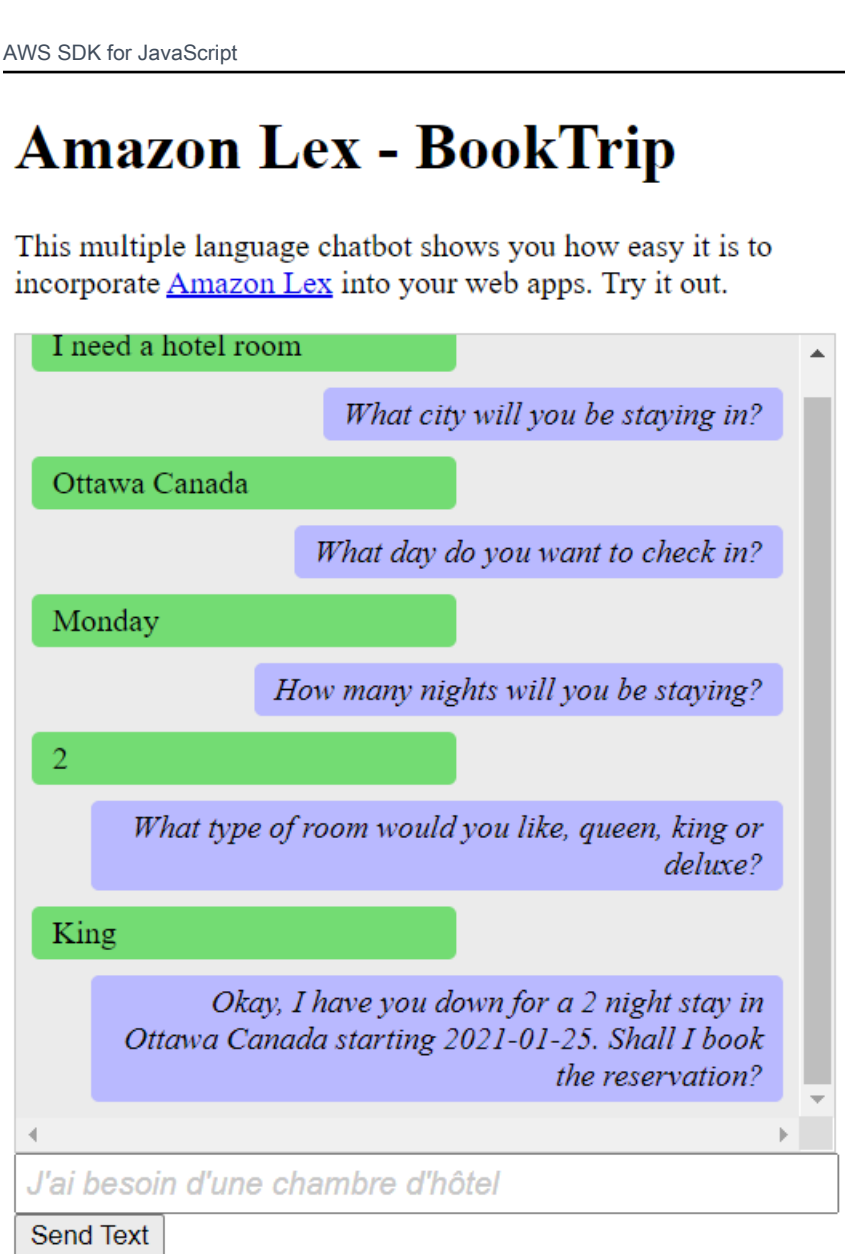
리소스 삭제

축하합니다! 를 사용하여 CloudWatch Amazon 예약 이벤트를 통해 Lambda 함수를 호출했습니다. AWS SDK for JavaScript 이 자습서의 시작 부분에서 설명한 것처럼 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다. 다음과 같이 이 자습서의 AWS 리소스 생성 주제에서 생성한 AWS CloudFormation 스택을 삭제하면 됩니다.

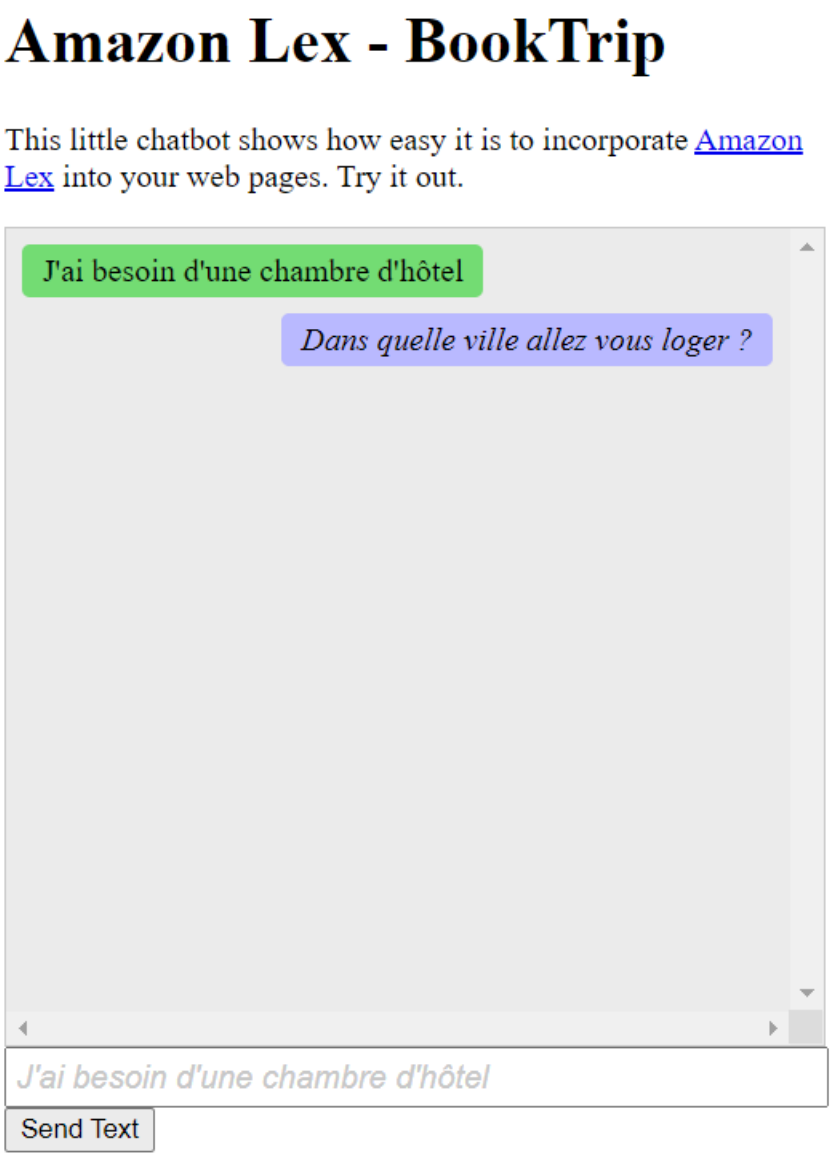
1. AWS CloudFormation 콘솔을 엽니다.
2. 스택 페이지에서 스택을 선택합니다.
3. 삭제(Delete)를 선택합니다.

Amazon Lex 챗봇 빌드

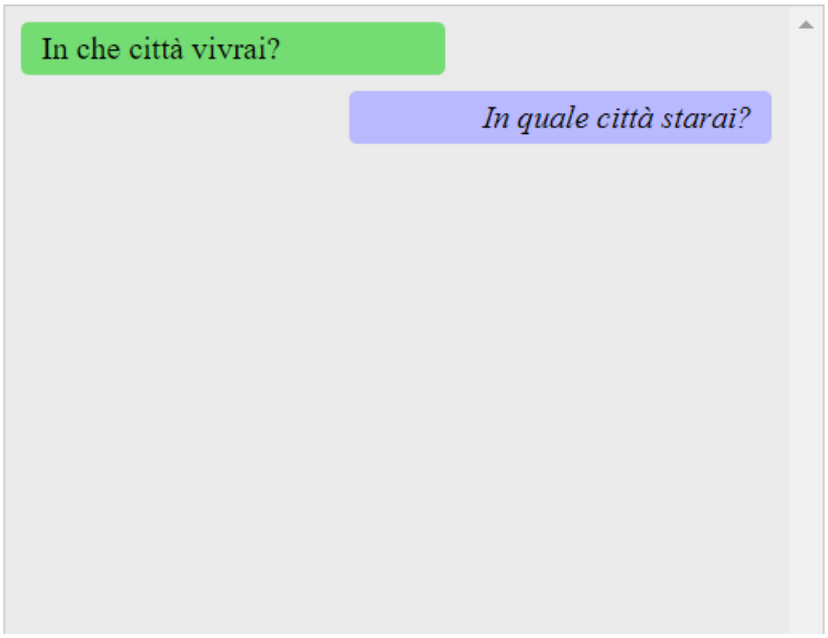
웹 애플리케이션 내에 Amazon Lex 챗봇을 생성하여 웹 사이트 방문자의 참여를 유도할 수 있습니다. Amazon Lex 챗봇은 사람과 직접 접촉하지 않고도 사용자와 온라인 채팅 대화를 수행하는 기능입니다. 예를 들어 다음 그림은 호텔 객실 예약과 관련하여 사용자의 참여를 유도하는 Amazon Lex 챗봇을 보여줍니다.



이 AWS 자습서에서 생성한 Amazon Lex 챗봇은 여러 언어를 처리할 수 있습니다. 예를 들어 프랑스어를 구사하는 사용자는 프랑스어 텍스트를 입력하고 프랑스어로 응답을 받을 수 있습니다.



마찬가지로, 사용자는 이탈리아어로 Amazon Lex 챗봇과 소통할 수 있습니다.



이 AWS 자습서에서는 Amazon Lex 챗봇을 생성하고 이를 Node.js 웹 애플리케이션에 통합하는 방법을 안내합니다. AWS SDK for JavaScript(v3) 는 다음 AWS 서비스를 호출하는 데 사용됩니다.

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

완료 비용: 이 문서에 포함된 AWS 서비스는 AWS 프리 티어에 포함됩니다.

참고: 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다.

앱을 빌드하려면 다음을 수행합니다.

1. 사전 조건
2. 리소스를 프로비저닝합니다.
3. Amazon Lex 챗봇 생성
4. HTML 생성

5. 브라우저 스크립트 생성
6. 다음 단계

필수 조건

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. 의 지침을 따르십시오 GitHub.
- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요.
> 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

AWS 리소스 생성

이 자습서를 시작하려면 다음 리소스가 필요합니다.

- 다음에 대한 권한이 연결된 미인증 IAM 역할:
  - Amazon Comprehend
  - Amazon Translate
  - Amazon Lex

이 리소스를 수동으로 생성할 수 있지만, 이 자습서에 설명된 대로 AWS CloudFormation을 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

AWS CloudFormation을 사용하여 AWS 리소스 생성

AWS CloudFormation을 사용하여 AWS 인프라 배포를 예상한 대로 반복해서 생성하고 프로비저닝할 수 있습니다. AWS CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

AWS CLI를 사용하여 AWS CloudFormation 스택을 생성하려면 다음을 수행합니다.

1. AWS CLI 사용 설명서의 지침에 따라 AWS CLI를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 `setup.yaml` 디렉터리에 이름이 지정된 파일을 만들고 여기에 콘텐츠를 복사합니다. GitHub

   > **Note**
   >
   > AWS CloudFormation템플릿은 여기에서 AWS CDK 사용할 수 있는 것을 사용하여 생성되었습니다 GitHub. AWS CDK에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 가이드 섹션을 참조하세요.

3. 명령줄에서 다음 명령을 실행하여 *STACK_NAME*을 스택의 고유한 이름으로 바꿉니다.

   > **Important**
   >
   > 스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

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

`create-stack` 명령 파라미터에 대한 자세한 내용은 AWS CLI 명령 참조 가이드 및 AWS CloudFormation 사용 설명서를 참조하세요.

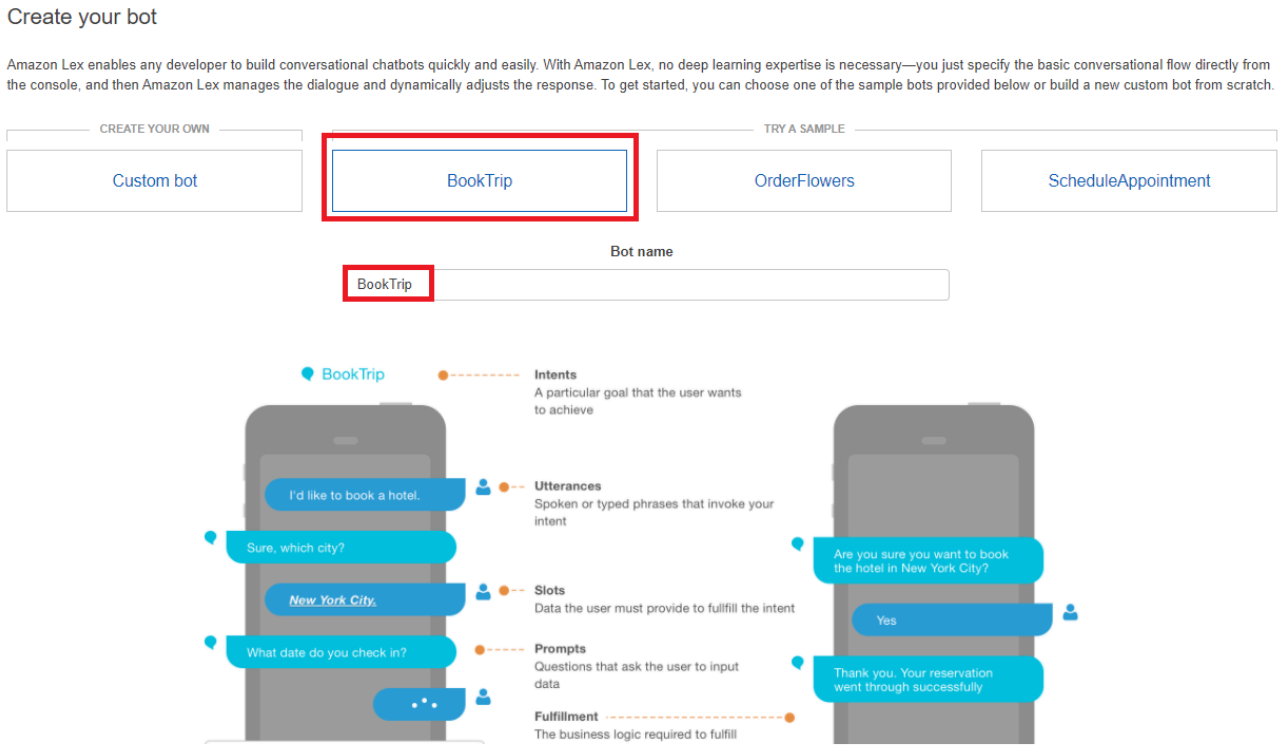
생성된 리소스를 보려면 Amazon Lex 콘솔을 열고 스택을 선택한 다음, 리소스 탭을 선택합니다.

Amazon Lex 봇 생성

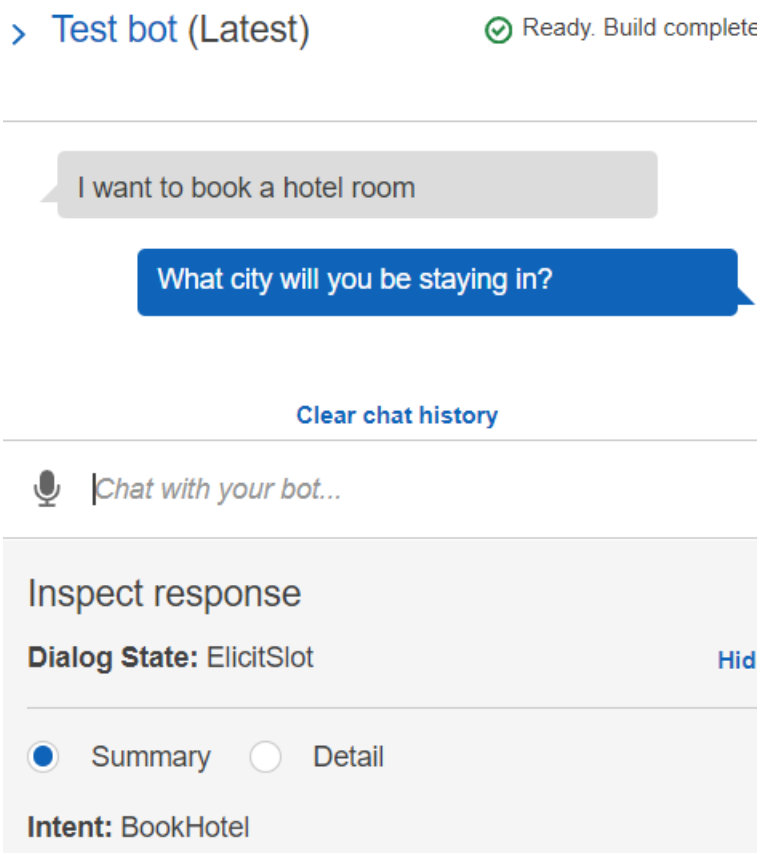
> **Important**
>
> Amazon Lex 콘솔 V1을 사용하여 봇을 생성합니다. 이 예는 V2를 사용하여 생성된 봇에서는 작동하지 않습니다.

첫 번째 단계는 Amazon Web Services Management Console을 사용하여 Amazon Lex 챗봇을 생성하는 것입니다. 이 예에서는 Amazon Lex BookTrip예제가 사용됩니다. 자세한 내용은 Book Trip 단원을 참조하세요.

- Amazon Web Services Management Console에 로그인하고 Amazon Web Services 콘솔에서 Amazon Lex 콘솔을 엽니다.
- 봇 페이지에서 생성을 선택합니다.
- BookTrip블루프린트를 선택합니다 (기본 봇 이름 유지 BookTrip).



- 기본 설정을 입력하고 [Create] 를 선택합니다 (콘솔에 BookTrip봇이 표시됨). 편집기 탭에서 미리 구성된 intent의 세부 정보를 검토합니다.
- 테스트 창에서 봇을 테스트합니다. 호텔 객실을 예약하고 싶어요를 입력해 테스트를 시작합니다.



- 게시를 선택하고 별칭 이름을 지정합니다(AWS SDK for JavaScript를 사용할 때 이 값이 필요함).

> **Note**
>
> JavaScript 코드에서 봇 이름과 봇 별칭을 참조해야 합니다.

HTML 생성

`index.html`이라는 이름의 파일을 만듭니다. 아래 코드를 복사하여 `index.html`에 붙여 넣습니다. 이 HTML은 `main.js`를 참조합니다. 이는 필수 AWS SDK for JavaScript 모듈이 포함된 index.js의 번들 버전입니다. HTML 생성에서 이 파일을 생성합니다. 또한 `index.html`은 스타일을 추가하는 `style.css`도 참조합니다.

```
<!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>
```

이 코드는 여기에서도 확인할 수 있습니다. GitHub

브라우저 스크립트 생성

`index.js`이라는 이름의 파일을 만듭니다. 아래 코드를 복사하여 `index.js`에 붙여 넣습니다. 필수 AWS SDK for JavaScript 모듈과 명령을 가져옵니다. Amazon Lex, Amazon Comprehend 및 Amazon Translate용 클라이언트를 생성합니다. *`REGION`*을 AWS 리전으로 바꾸고, *`IDENTITY_POOL_ID`*를 AWS 리소스 생성 에서 생성한 자격 증명 풀 ID로 바꿉니다. 이 자격 증명 풀 ID를 검색하려면 Amazon Cognito 콘솔에서 자격 증명 풀을 열고 자격 증명 풀 편집을 선택한 다음, 측면 메뉴에서 샘플 코드를 선택합니다. 자격 증명 풀 ID는 콘솔에 빨간색 텍스트로 표시됩니다.

먼저, `libs` 디렉터리를 생성하고, `comprehendClient.js`, `lexClient.js`, `translateClient.js`라는 세 가지 파일을 생성하여 필수 서비스 클라이언트 객체를 생성합니다. 아래의 적절한 코드를 각 파일에 붙여 넣고 각 파일의 *`REGION`* 및 *`IDENTITY_POOL_ID`*를 바꿉니다.

> **Note**
>
> AWS CloudFormation을 사용하여 AWS 리소스 생성에서 생성한 Amazon Cognito 자격 증명 풀 ID를 사용하세요.

```
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 };
```

이 코드는 여기에서 확인할 수 있습니다 GitHub. .

다음으로, `index.js` 파일을 생성하고 아래 코드를 이 파일에 붙여 넣습니다.

BOT_ALIAS 및 *BOT_NAME*을 각각 Amazon Lex 봇의 별칭 및 이름으로 바꾸고, *USER_ID*를 사용자 ID로 바꿉니다. `createResponse` 비동기 함수는 다음을 수행합니다.

- 사용자가 입력한 텍스트를 브라우저에 가져오고 Amazon Comprehend를 사용하여 언어 코드를 결정합니다.
- 언어 코드를 가져오고 Amazon Translate를 사용하여 텍스트를 영어로 번역합니다.
- 번역된 텍스트를 가져오고 Amazon Lex를 사용하여 응답을 생성합니다.
- 브라우저 페이지에 응답을 게시합니다.

```
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 loadNewItems() {
  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;
```

이 코드는 여기에서 확인할 수 있습니다 GitHub. .

이제 webpack을 사용하여 `index.js` 및 AWS SDK for JavaScript 모듈을 단일 파일인 `main.js`로 번들링합니다.

1. webpack을 아직 설치하지 않았다면 이 예의 필수 조건에 따라 설치합니다.

   > **Note**
   >
   > Webpack에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

2. 명령줄에서 다음을 실행하여 이 예제의 JavaScript 내용을 라는 파일로 묶습니다`main.js`.

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

다음 단계

축하합니다! Amazon Lex를 사용하여 대화형 사용자 환경을 생성하는 Node.js 애플리케이션을 생성했습니다. 이 자습서의 시작 부분에서 설명한 것처럼 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다. 다음과 같이 이 자습서의 AWS 리소스 생성 주제에서 생성한 AWS CloudFormation 스택을 삭제하면 됩니다.

1. AWS CloudFormation 콘솔을 엽니다.
2. 스택 페이지에서 스택을 선택합니다.
3. 삭제을 선택합니다.

더 많은 AWS 교차 서비스 예를 확인하려면 AWS SDK for JavaScript cross-service examples를 참조하세요.

메시징 애플리케이션 예 생성

AWS SDK for JavaScript 및 Amazon Simple Queue Service(Amazon SQS)를 사용하여 메시지를 보내고 검색하는 AWS 애플리케이션을 생성할 수 있습니다. 메시지는 메시지의 순서가 일관되게 유지되도록 보장하는 선입선출(FIFO) 방식 대기열에 저장됩니다. 예를 들어 대기열에 저장된 첫 번째 메시지는 대기열에서 읽는 첫 번째 메시지입니다.

> **Note**
>
> Amazon SQS에 관한 자세한 내용은 What is Amazon Simple Queue Service? 단원을 참조하세요.

이 자습서에서는 AWS Messaging이라는 Node.js 애플리케이션을 생성합니다.

완료 비용: 이 문서에 포함된 AWS 서비스는 AWS 프리 티어에 포함됩니다.

참고: 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다.

앱을 빌드하려면 다음을 수행합니다.

1. 사전 조건
2. 리소스를 프로비저닝합니다.
3. 워크플로 이해
4. HTML 생성
5. 브라우저 스크립트 생성
6. 다음 단계

필수 조건

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 Node TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. 의 지침을 따르십시오 GitHub.

- 사용자 자격 증명을 사용하여 공유 구성 파일을 생성합니다. 공유 보안 인증 파일 제공에 관한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 Shared config and credentials files 단원을 참조하세요.

> **Important**
>
> 이 예에서는 ECMAScript6(ES6)를 사용합니다. 따라서 Node.js 버전 13.x 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 Node.js downloads를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 JavaScript ES6/CommonJS 구문 단원을 참조하세요.

AWS 리소스 생성

이 자습서를 시작하려면 다음 리소스가 필요합니다.

- Amazon SQS에 대한 권한이 있는 미인증 IAM 역할.
- Message.fifo라는 FIFO Amazon SQS 대기열 - 대기열 생성에 관한 자세한 내용은 Creating an Amazon SQS queue 단원을 참조하세요.

이 리소스를 수동으로 생성할 수 있지만, 이 자습서에 설명된 대로 AWS CloudFormation(AWS CloudFormation)을 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

> **Note**
>
> AWS CloudFormation은 클라우드 애플리케이션 리소스를 정의할 수 있게 지원하는 소프트웨어 개발 프레임워크입니다. 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하십시오.

AWS CloudFormation을 사용하여 AWS 리소스 생성

AWS CloudFormation을 사용하여 AWS 인프라 배포를 예상한 대로 반복해서 생성하고 프로비저닝할 수 있습니다. AWS CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

AWS CLI를 사용하여 AWS CloudFormation 스택을 생성하려면 다음을 수행합니다.

1. AWS CLI 사용 설명서의 지침에 따라 AWS CLI를 설치하고 구성합니다.

2. 프로젝트 폴더의 루트 `setup.yaml` 디렉터리에 이름이 지정된 파일을 만들고 여기에 콘텐츠를 복사합니다. GitHub

> **Note**
>
> AWS CloudFormation템플릿은 여기에서 AWS CDK 사용할 수 있는 것을 사용하여 생성되었습니다 GitHub. AWS CDK에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 가이드 섹션을 참조하세요.

3. 명령줄에서 다음 명령을 실행하여 *STACK_NAME*을 스택의 고유한 이름으로 바꿉니다.

> **Important**
>
> 스택 이름은 AWS 리전 및 AWS 계정 내에서 고유해야 합니다. 최대 128자까지 지정할 수 있으며 숫자와 하이픈을 사용할 수 있습니다.

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

`create-stack` 명령 파라미터에 대한 자세한 내용은 AWS CLI 명령 참조 가이드 및 AWS CloudFormation 사용 설명서를 참조하세요.

생성된 리소스를 보려면 AWS 관리 콘솔에서 AWS CloudFormation을 열고 스택을 선택한 다음, 리소스 탭을 선택합니다.

AWS Messaging 애플리케이션 이해

SQS 대기열에 메시지를 보내려면 애플리케이션에 메시지를 입력하고 전송을 선택합니다.

메시지가 전송되면 애플리케이션에 메시지가 표시됩니다.

제거를 선택하여 Amazon SQS 대기열에서 메시지를 제거할 수 있습니다. 그러면 대기열이 비어 있게 되고 애플리케이션에 메시지가 표시되지 않습니다.

다음은 애플리케이션이 메시지를 처리하는 방법을 설명합니다.

- 사용자가 자신의 이름을 선택하고 메시지를 입력한 다음, 메시지를 제출합니다. 그러면 `pushMessage` 함수가 시작됩니다.

- pushMessage는 Amazon SQS 대기열 URL을 검색한 다음, 고유한 메시지 ID 값(GUID), 메시지 텍스트 및 사용자가 포함된 메시지를 Amazon SQS 대기열에 보냅니다.
- pushMessage는 Amazon SQS 대기열에서 메시지를 검색하고 각 메시지의 사용자와 메시지를 추출하며 메시지를 표시합니다.
- 사용자는 메시지를 제거하여 Amazon SQS 대기열 및 사용자 인터페이스에서 메시지를 삭제할 수 있습니다.

HTML 페이지 생성

이제 애플리케이션의 그래픽 사용자 인터페이스(GUI)에 필요한 HTML 파일을 생성합니다. index.html이라는 이름의 파일을 만듭니다. 아래 코드를 복사하여 index.html에 붙여 넣습니다. 이 HTML은 main.js를 참조합니다. 이는 필수 AWS SDK for JavaScript 모듈이 포함된 index.js의 번들 버전입니다.

```
<!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">
          <img
            src="../public/images/av2.png"
            alt="Avatar"
            class="right"
            style="width: 100%"
          />
          <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>
```

이 코드는 여기에서도 확인할 수 GitHub 있습니다.

브라우저 스크립트 생성

이 항목에서는 앱의 브라우저 스크립트를 생성합니다. 브라우저 스크립트를 생성한 경우 번들링 JavaScript에 설명된 대로 `main.js`라는 파일로 번들링합니다.

`index.js`이라는 이름의 파일을 만듭니다. 여기서부터 코드를 복사해서 붙여넣으세요. GitHub

이 코드는 다음 단원에서 설명합니다.

1. 구성
2. populateChat
3. pushmessages
4. purge

구성

먼저, `libs` 디렉터리를 생성하고, `sqsClient.js`라는 파일을 생성하여 필수 Amazon SQS 클라이언트 객체를 생성합니다. 각각에서 *`REGION`* 및 *`IDENTITY_POOL_ID`*를 바꿉니다.

> **Note**
>
> AWS 리소스 생성 에서 생성한 Amazon Cognito 자격 증명 풀 ID를 사용하세요.

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

`index.js`에서, 필수 AWS SDK for JavaScript 모듈과 명령을 가져옵니다. *`SQS_QUEUE_NAME`*을 AWS 리소스 생성 에서 생성한 Amazon SQS 대기열 이름으로 바꿉니다.

```
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

populateChat 함수 onload는 Amazon SQS 대기열의 URL을 자동으로 검색하고 대기열의 모든 메시지를 검색하여 표시합니다.

```
$(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 =
          '<img src="' +
          image_url +
          '" alt="Avatar" class="right" style=""width:100%;"">';
        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);
  }
};
```

메시지 푸시

사용자가 자신의 이름을 선택하고 메시지를 입력한 다음, 메시지를 제출합니다. 그러면 pushMessage 함수가 시작됩니다. pushMessage는 Amazon SQS 대기열 URL을 검색한 다음, 고유한 메시지 ID 값

(GUID), 메시지 텍스트 및 사용자가 포함된 메시지를 Amazon SQS 대기열에 보냅니다. 그런 다음, Amazon SQS 대기열에서 모든 메시지를 검색하여 표시합니다.

```
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 =
        '<img src="' +
        image_url +
        '" alt="Avatar" class="right" style=""width:100%;"">';
      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;
```

메시지 삭제

purge는 Amazon SQS 대기열 및 사용자 인터페이스에서 메시지를 삭제합니다.

```
// 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;
```

번들링 JavaScript

이 전체 브라우저 스크립트 코드는 여기에서 확인할 수 있습니다. GitHub .

이제 webpack을 사용하여 index.js 및 AWS SDK for JavaScript 모듈을 단일 파일인 main.js로 번들링합니다.

1. webpack을 아직 설치하지 않았다면 이 예의 필수 조건에 따라 설치합니다.

> **Note**
>
> Webpack에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

2. 명령줄에서 다음을 실행하여 이 예제의 JavaScript 내용을 라는 파일로 묶습니다<index.js>.

```
webpack index.js --mode development --target web --devtool false -o main.js
```

다음 단계

축하합니다! Amazon SQS를 사용하는 AWS Messaging 애플리케이션을 생성하고 배포했습니다. 이 자습서의 시작 부분에서 설명한 것처럼 더 이상 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다. 다음과 같이 이 자습서의 AWS 리소스 생성 주제에서 생성한 AWS CloudFormation 스택을 삭제하면 됩니다.

1. AWS 관리 콘솔에서 AWS CloudFormation을 엽니다.
2. 스택 페이지를 열고 스택을 선택합니다.
3. 삭제(Delete)를 선택합니다.

AWS Cloud9 와 함께 사용 AWS SDK for JavaScript

AWS Cloud9 와 함께 사용하면 JavaScript 브라우저만으로 브라우저 코드를 작성 및 실행할 수 있을 뿐만 아니라 Node.js 코드를 작성, 실행 및 디버그할 수도 있습니다. AWS SDK for JavaScript AWS Cloud9 코드 편집기 및 터미널과 같은 도구와 Node.js 코드용 디버거가 포함되어 있습니다.

AWS Cloud9 IDE는 클라우드 기반이므로 인터넷이 연결된 컴퓨터를 사용하여 사무실, 집 또는 어디서나 프로젝트 작업을 수행할 수 있습니다. 에 대한 일반 정보는 사용 AWS Cloud9설명서를 참조하십시오.AWS Cloud9

다음 단계는 에 대한 JavaScript SDK를 AWS Cloud9 사용하여 설정하는 방법을 설명합니다.

목차

1단계: 사용할 AWS 계정 설정 AWS Cloud9

계정에 대한 액세스 권한을 가진 AWS Identity and Access Management (IAM) 개체 (예: IAM 사용자)로 AWS Cloud9 콘솔에 AWS Cloud9 로그인하여 사용을 AWS Cloud9 시작하십시오. AWS

AWS 계정에 IAM 엔티티를 설정하여 AWS Cloud9 콘솔에 AWS Cloud9액세스하고 로그인하려면 사용 설명서의 팀 설정을 참조하십시오 AWS Cloud9.AWS Cloud9

2단계: AWS Cloud9 개발 환경 설정

AWS Cloud9 콘솔에 로그인한 후 콘솔을 사용하여 AWS Cloud9 개발 환경을 생성합니다. 환경을 만든 후 해당 환경의 IDE를 AWS Cloud9 엽니다.

자세한 내용은 AWS Cloud9 사용 설명서의 Creating an environment in AWS Cloud9 단원을 참조하세요.

> **Note**
>
> 콘솔에서 처음으로 환경을 생성할 때 Create a new instance for environment (EC2)(환경에 대한 새 인스턴스 생성(EC2)) 옵션을 선택하는 것이 좋습니다. 이 옵션은 환경을 만들고, Amazon EC2 인스턴스를 시작한 다음, 새 인스턴스를 새 환경에 AWS Cloud9 연결하도록 지시합니다. 이것이 사용을 AWS Cloud9시작하는 가장 빠른 방법입니다.

3단계: SDK 설정 대상 JavaScript

개발 환경용 IDE를 연 후 AWS Cloud9 다음 절차 중 하나 또는 모두에 따라 IDE를 사용하여 환경에 적합한 JavaScript SDK를 설정하십시오.

Node.js 버전을 위한 JavaScript SDK를 설정하려면

1. 터미널이 IDE에 아직 열려 있지 않은 경우 터미널을 엽니다. 이렇게 하려면 IDE의 메뉴 모음에서 Window, New Terminal(창, 새 터미널)을 선택합니다.
2. 다음 명령을 실행하여 SDK의 `Cloud9` 클라이언트를 설치하는 npm 데 사용합니다. JavaScript

   ```
   npm install @aws-sdk/client-cloud9
   ```

   IDE에서 npm을 찾지 못하면 아래 순서에 따라 한 번에 하나씩 다음 명령을 실행해 npm을 설치합니다. 이러한 명령은 이 항목의 앞 부분에서 Create a new instance for environment (EC2)(환경에 대한 새 인스턴스 생성(EC2)) 옵션을 선택한 것으로 가정합니다.

   > **Warning**
   >
   > AWS 다음 코드는 제어하지 않습니다. 실행하기 전에 먼저 신뢰성과 무결성을 확인해야 합니다. 이 코드에 대한 자세한 내용은 nvm (노드 버전 관리자) GitHub 저장소에서 찾을 수 있습니다.

   ```
   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).
```

JavaScript 브라우저에서 SDK를 설정하려면

HTML 페이지에서 SDK를 JavaScript WebPack 사용하려면 필요한 클라이언트 모듈과 모든 필수 JavaScript 함수를 단일 JavaScript 파일로 묶고 HTML 페이지의 스크립트 태그에 추가하는 데 사용합니다. <head> 예:

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

> **Note**
>
> Webpack에 관한 자세한 내용은 웹팩이 포함된 번들 애플리케이션 단원을 참조하세요.

4단계: 코드 예 다운로드

이전 단계에서 연 터미널을 사용하여 AWS Cloud9 개발 환경에 SDK용 예제 코드를 JavaScript 다운로드합니다. IDE에서 터미널이 아직 열려 있지 않은 경우 IDE의 메뉴 모음에서 Window, New Terminal(창, 새 터미널)을 선택하여 엽니다.

예제 코드를 다운로드하려면 다음 명령을 실행합니다. 이 명령은 공식 AWS SDK 설명서에 사용된 모든 코드 예제의 사본을 사용자 환경의 루트 디렉터리로 다운로드합니다.

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

SDK의 코드 예제를 찾으려면 환경 창을 사용하여 를 여십시오. 여기서 *ENVIRONMENT_NAME* *# ## ###* 이름입니다. JavaScript *ENVIRONMENT_NAME*\aws-doc-sdk-examples \javascriptv3\example_code/src AWS Cloud9

이러한 코드 예제와 다른 코드 예제를 사용하는 방법을 알아보려면 SDK에서 코드 예제를 참조하십시오. JavaScript

5단계: 코드 예 실행 및 디버깅

AWS Cloud9 개발 환경에서 코드를 실행하려면 AWS Cloud9 사용 설명서에서 코드 실행을 참조하십시오.

Node.js 코드를 디버깅하려면 AWS Cloud9 사용 설명서의 코드 디버그 단원을 참조하세요.

JavaScript (v3) 용 SDK 코드 예제

이 항목의 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하는 방법을 보여줍니다. AWS

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

교차 서비스 예시는 여러 AWS 서비스전반에서 작동하는 샘플 애플리케이션입니다.

예

- JavaScript (v3) 용 SDK를 사용한 작업 및 시나리오
- JavaScript (v3) 용 SDK를 사용한 크로스 서비스 예제

JavaScript (v3) 용 SDK를 사용한 작업 및 시나리오

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. AWS 서비스

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

서비스

- JavaScript (v3) 용 SDK를 사용한 Auto Scaling 예제
- (v3) 용 JavaScript SDK를 사용하는 아마존 베드락 예제
- (v3) 용 JavaScript SDK를 사용하는 Amazon 베드락 런타임 예제
- (v3) 용 SDK를 사용하는 아마존 베드락용 JavaScript 에이전트 예제
- (v3) 용 SDK를 사용하는 Amazon 베드락 에이전트 런타임 예제 JavaScript

- CloudWatch JavaScript (v3) 용 SDK 사용 예제
- CloudWatch JavaScript (v3) 용 SDK를 사용한 이벤트 예제
- CloudWatch JavaScript (v3) 용 SDK를 사용한 로그 예제
- CodeBuild JavaScript (v3) 용 SDK 사용 예제
- (v3) 에 JavaScript SDK를 사용하는 Amazon Cognito 자격 증명 공급자 예제
- (v3) 에 JavaScript SDK를 사용하는 DynamoDB 예제
- (v3) 용 JavaScript SDK를 사용하는 Amazon EC2 예제
- JavaScript (v3) 용 SDK를 사용한 Elastic Load Balancing 예제
- EventBridge JavaScript (v3) 용 SDK 사용 예제
- AWS Glue JavaScript (v3) 용 SDK 사용 예제
- HealthImaging JavaScript (v3) 용 SDK 사용 예제
- JavaScript (v3) 용 SDK를 사용한 IAM 예제
- (v3) 에 JavaScript SDK를 사용하는 Lambda 예제
- (v3) 용 JavaScript SDK를 사용한 Amazon Personalize 예제
- (v3) 용 JavaScript SDK를 사용한 Amazon Personalize 이벤트 예제
- (v3) 용 JavaScript SDK를 사용한 Amazon Personalize 런타임 예제
- (v3) 용 JavaScript SDK를 사용한 Amazon Pinpoint 예제
- (v3) 용 JavaScript SDK를 사용하는 Amazon Redshift 예제
- JavaScript (v3) 용 SDK를 사용하는 Amazon S3 예제
- (v3) 용 JavaScript SDK를 사용한 S3 글레이셔 예제
- SageMaker JavaScript (v3) 용 SDK 사용 예제
- JavaScript (v3) 용 SDK를 사용한 Secrets Manager 예제
- JavaScript (v3) 용 SDK를 사용하는 Amazon SES 예제
- JavaScript (v3) 에 SDK를 사용하는 Amazon SNS 예제
- (v3) 용 JavaScript SDK를 사용하는 Amazon SQS 예제
- JavaScript (v3) 용 SDK를 사용한 Step Functions 예제
- AWS STS JavaScript (v3) 용 SDK 사용 예제
- AWS Support JavaScript (v3) 용 SDK 사용 예제
- (v3) 용 JavaScript SDK를 사용한 Amazon Transcribe 예제

JavaScript (v3) 용 SDK를 사용한 Auto Scaling 예제

다음 코드 예제는 Auto Scaling과 함께 AWS SDK for JavaScript (v3) 를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 상황에 맞게 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업
- 시나리오

작업

오토 스케일링 그룹에 ELB 대상 그룹 연결

다음 코드 예제에서는 오토 스케일링 그룹에 ELB 대상 그룹을 연결하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const client = new AutoScalingClient({});
await client.send(
  new AttachLoadBalancerTargetGroupsCommand({
    AutoScalingGroupName: NAMES.autoScalingGroupName,
    TargetGroupARNs: [state.targetGroupArn],
  }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API AttachLoadBalancerTargetGroups참조를 참조하십시오.

시나리오

복원력이 뛰어난 서비스 구축 및 관리

다음 코드 예제에서는 책, 영화, 노래 추천을 반환하는 로드 밸런싱 웹 서비스를 만드는 방법을 보여줍니다. 이 예제에서는 서비스가 장애에 대응하는 방법과 장애 발생 시 복원력을 높이기 위해 서비스를 재구성하는 방법을 보여줍니다.

- Amazon EC2 Auto Scaling 그룹을 사용하여 시작 템플릿을 기반으로 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스를 생성하고 인스턴스 수를 지정된 범위 내로 유지합니다.
- Elastic Load Balancing으로 HTTP 요청을 처리하고 배포합니다.
- 오토 스케일링의 인스턴스 상태를 모니터링하고 요청을 정상 인스턴스로만 전달합니다.
- 각 EC2 인스턴스에서 Python 웹 서버를 실행하여 HTTP 요청을 처리합니다. 웹 서버는 추천 및 상태 확인으로 응답합니다.
- Amazon DynamoDB 테이블을 사용하여 추천 서비스를 시뮬레이션합니다.
- AWS Systems Manager 매개변수를 업데이트하여 요청 및 상태 확인에 대한 웹 서버 응답을 제어합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
#!/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);
}
```

모든 리소스를 배포하기 위한 단계를 생성합니다.

```
// 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,
      );
    }
  }),
];
```

데모를 실행하기 위한 단계를 생성합니다.

```
// 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;
}
```

모든 리소스를 폐기하는 단계를 생성합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - AttachLoadBalancerTargetGroups

- CreateAutoScalingGroup
- CreateInstanceProfile
- CreateLaunchTemplate
- CreateListener
- CreateLoadBalancer
- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

(v3) 용 JavaScript SDK를 사용하는 아마존 베드락 예제

다음 코드 예제는 Amazon Bedrock과 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

시작하기

Amazon Bedrock 시작

다음 코드 예시에서는 Amazon Bedrock 사용을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API ListFoundationModels참조를 참조하십시오.

주제

- 작업

작업

Amazon Bedrock 파운데이션 모델에 대한 세부 정보 가져오기

다음 코드 예시에서는 Amazon Bedrock 파운데이션 모델에 대한 세부 정보를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

파운데이션 모델에 대한 세부 정보를 가져옵니다.

```
// 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);
}
```

- API 세부 정보는 AWS SDK for JavaScript API GetFoundationModel참조를 참조하십시오.

사용 가능한 Amazon Bedrock 기본 모델 나열

다음 코드 예제에서는 사용 가능한 Amazon Bedrock 기반 모델을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

사용 가능한 파운데이션 모델을 나열합니다.

```
// 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);
}
```

- API 세부 정보는 AWS SDK for JavaScript API ListFoundationModels참조를 참조하십시오.

(v3) 용 JavaScript SDK를 사용하는 Amazon 베드락 런타임 예제

다음 코드 예제는 Amazon Bedrock Runtime과 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

작업

AI21 Labs Jurassic-2를 사용한 텍스트 생성

다음 코드 예시에서는 텍스트 생성을 위해 Amazon Bedrock에서 AI21 Labs Jurassic-2 모델을 간접 호출하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

AI21 Labs Jurassic-2 파운데이션 모델을 간접 호출하여 텍스트를 생성합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API InvokeModel참조를 참조하십시오.

아마존 타이탄 텍스트 G1을 사용한 텍스트 생성

다음 코드 예제는 텍스트 생성을 위해 Amazon Bedrock에서 Amazon Titan Text G1 모델을 호출하는 방법을 보여줍니다.

(v3) 용 SDK JavaScript

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon Titan Text G1 기반 모델을 호출하여 텍스트를 생성합니다.

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

- API 세부 정보는 API 참조를 참조하십시오 InvokeModel.AWS SDK for JavaScript

Anthropic Claude 2를 사용한 텍스트 생성

다음 코드 예시에서는 텍스트 생성을 위해 Amazon Bedrock에서 Anthropic Claude 모델을 간접 호출하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Anthropic Claude 2 파운데이션 모델을 간접 호출하여 텍스트를 생성합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API InvokeModel참조를 참조하십시오.

Meta Llama 2 Chat을 통한 텍스트 생성

다음 코드 예시에서는 텍스트 생성을 위해 Amazon Bedrock에서 Meta Llama 2 Chat 모델을 간접 호출하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Meta Llama 2 Chat 파운데이션 모델을 간접 호출하여 텍스트를 생성합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API InvokeModel참조를 참조하십시오.

Mistral 7B를 사용한 텍스트 생성

다음 코드 예제는 텍스트 생성을 위해 Amazon Bedrock에서 Mistral 7B 모델 모델을 호출하는 방법을 보여줍니다.

(JavaScript v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Mistral 7B 기반 모델을 호출하여 텍스트를 생성하십시오.

```
// 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 = `<s>[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");
  });
}
```

- API 세부 정보는 API 레퍼런스를 참조하십시오 InvokeModel.AWS SDK for JavaScript

믹스트럴 8x7B를 사용한 텍스트 생성

다음 코드 예제는 텍스트 생성을 위해 Amazon Bedrock에서 Mixtral 8x7B 모델 모델을 호출하는 방법을 보여줍니다.

(JavaScript v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Mixtral 8x7B 기반 모델을 호출하여 텍스트를 생성하십시오.

```
// 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 = `<s>[INST] ${prompt} [/INST]`;

  const modelId = "mistral.mixtral-8x7b-instruct-v0:1";

  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: Mixtral 8x7B");
  console.log(`Prompt: ${prompt}`);

  const completions = await invokeMistral7B(prompt);
  completions.forEach((completion) => {
    console.log("Completion:");
    console.log(completion);
    console.log("\n");
  });
}
```

- API 세부 정보는 API 레퍼런스를 참조하십시오. InvokeModelAWS SDK for JavaScript

(v3) 용 SDK를 사용하는 아마존 베드락용 JavaScript 에이전트 예제

다음 코드 예제는 Amazon Bedrock용 에이전트와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 상황에 GitHub 맞게 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

시작하기

안녕하세요. 아마존 베드락 에이전트

다음 코드 예제는 Amazon Bedrock용 에이전트 사용을 시작하는 방법을 보여줍니다.

(v3) 용 JavaScript SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - GetAgent
  - ListAgents

주제

- 작업

작업

에이전트를 생성합니다.

다음 코드 예시에서는 Amazon Bedrock 에이전트를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

에이전트를 생성합니다.

```
// 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);
}
```

- API 세부 정보는 AWS SDK for JavaScript API CreateAgent참조를 참조하십시오.

에이전트 삭제

다음 코드 예시에서는 Amazon Bedrock 에이전트를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

에이전트를 삭제합니다.

```
// 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);
}
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteAgent참조를 참조하십시오.

에이전트 정보 가져오기

다음 코드 예시에서는 Amazon Bedrock 에이전트에 대한 정보를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

에이전트를 가져옵니다.

```
// 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);
}
```

- API 세부 정보는 AWS SDK for JavaScript API GetAgent참조를 참조하십시오.

에이전트의 작업 그룹 나열

다음 코드 예시에서는 Amazon Bedrock 에이전트의 작업 그룹을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

에이전트의 작업 그룹을 나열합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API ListAgentActionGroups참조를 참조하십시오.

사용 가능한 에이전트를 나열하세요.

다음 코드 예시에서는 계정에 속한 Amazon Bedrock용 에이전트를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

계정에 속한 에이전트를 나열합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API ListAgents참조를 참조하십시오.

(v3) 용 SDK를 사용하는 Amazon 베드락 에이전트 런타임 예제 JavaScript

다음 코드 예제는 Amazon Bedrock Runtime용 에이전트와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

- 작업

작업

에이전트 간접 호출

다음 코드 예시에서는 Amazon Bedrock 에이전트를 간접 호출하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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);
}
```

- API 세부 정보는 AWS SDK for JavaScript API InvokeAgent참조를 참조하십시오.

CloudWatch JavaScript (v3) 용 SDK 사용 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. CloudWatch

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업

작업

지표 알람 생성

다음 코드 예제는 Amazon CloudWatch 경보를 생성 또는 업데이트하고 이를 지정된 지표, 지표 수학 식, 예외 항목 탐지 모델 또는 Metrics Insights 쿼리와 연결하는 방법을 보여줍니다.

(v3) 용 JavaScript SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutMetricAlarm참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutMetricAlarm참조를 참조하십시오.

경보 삭제

다음 코드 예제는 Amazon CloudWatch 경보를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteAlarms참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteAlarms참조를 참조하십시오.

지표에 대한 경보 설명

다음 코드 예제는 지표에 대한 Amazon CloudWatch 경보를 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DescribeAlarmsForMetric참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DescribeAlarmsForMetric참조를 참조하십시오.

경보 작업 비활성화

다음 코드 예제는 Amazon CloudWatch 알람 작업을 비활성화하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DisableAlarmActions참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DisableAlarmActions참조를 참조하십시오.

경보 작업 활성화

다음 코드 예제는 Amazon CloudWatch 알람 작업을 활성화하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API EnableAlarmActions참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API EnableAlarmActions참조를 참조하십시오.

지표 나열

다음 코드 예제는 Amazon CloudWatch 지표의 메타데이터를 나열하는 방법을 보여줍니다. 지표에 대한 데이터를 가져오려면 GetMetricData 또는 GetMetricStatistics 작업을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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);
};
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListMetrics참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListMetrics참조를 참조하십시오.

지표에 데이터 추가

다음 코드 예제는 지표 데이터 포인트를 Amazon에 게시하는 방법을 보여줍니다 CloudWatch.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutMetricData참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutMetricData참조를 참조하십시오.

CloudWatch JavaScript (v3) 용 SDK를 사용한 이벤트 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) with CloudWatch Events를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업

작업

대상 추가

다음 코드 예제는 Amazon CloudWatch Events 이벤트에 대상을 추가하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutTargets참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutTargets참조를 참조하십시오.

예약된 규칙 생성

다음 코드 예제는 Amazon CloudWatch Events 예약 규칙을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutRule참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutRule참조를 참조하십시오.

이벤트 전송

다음 코드 예제는 Amazon CloudWatch Events 이벤트를 전송하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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();
```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutEvents참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutEvents참조를 참조하십시오.

CloudWatch JavaScript (v3) 용 SDK를 사용한 로그 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) with CloudWatch Logs를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업
- 시나리오

작업

로그 그룹 생성

다음 코드 예제는 새 로그 CloudWatch 로그 그룹을 만드는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateLogGroup참조를 참조하십시오.

구독 필터 생성

다음 코드 예제는 Amazon CloudWatch Logs 구독 필터를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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();
```

- API 세부 정보는 AWS SDK for JavaScript API PutSubscriptionFilter참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

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

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

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutSubscriptionFilter참조를 참조하십시오.

로그 그룹 삭제

다음 코드 예제는 기존 로그 CloudWatch 로그 그룹을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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();
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteLogGroup참조를 참조하십시오.

구독 필터 삭제

다음 코드 예제는 Amazon CloudWatch Logs 구독 필터를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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();
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteSubscriptionFilter참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

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

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

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteSubscriptionFilter참조를 참조하십시오.

기존 구독 필터 설명

다음 코드 예제는 Amazon CloudWatch Logs 기존 구독 필터를 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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();
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeSubscriptionFilters참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

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

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

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

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DescribeSubscriptionFilters참조를 참조하십시오.

로그 그룹 설명

다음 코드 예제는 로그 CloudWatch 로그 그룹을 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeLogGroups참조를 참조하십시오.

쿼리 결과 가져오기

다음 코드 예시에서는 쿼리 결과를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
  /**
   * Simple wrapper for the GetQueryResultsCommand.
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }
```

- API 세부 정보는 AWS SDK for JavaScript API GetQueryResults참조를 참조하십시오.

Live Tail 세션 시작

다음 코드 예시에서는 기존 로그 그룹/로그 스트림에 대해 Live Tail 세션을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

필수 파일을 포함합니다.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-
cloudwatch-logs";
```

Live Tail 세션의 이벤트를 처리합니다.

```
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)
    }
}
```

Live Tail 세션을 시작합니다.

```
    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);
    }
```

일정 시간이 경과하면 Live Tail 세션을 중단합니다.

```
    /* Set a timeout to close the client. This will stop the Live Tail session. */
    setTimeout(function() {
        console.log("Client timeout");
        client.destroy();
      }, 10000);
```

- API에 대한 자세한 내용은 API StartLiveTail레퍼런스를 참조하십시오.AWS SDK for JavaScript

쿼리 시작

다음 코드 예시에서는 쿼리를 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 * 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;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API StartQuery참조를 참조하십시오.

시나리오

대용량 쿼리 실행

다음 코드 예제는 CloudWatch 로그를 사용하여 10,000개 이상의 레코드를 쿼리하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

진입점입니다.

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

필요한 경우 쿼리를 여러 단계로 분할하는 클래스입니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - GetQueryResults
  - StartQuery

CodeBuild JavaScript (v3) 용 SDK 사용 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. CodeBuild

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

작업

프로젝트 생성

다음 코드 예제는 CodeBuild 프로젝트를 만드는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

프로젝트 생성

```
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-xxxxxxx-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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateProject참조를 참조하십시오.

(v3) 에 JavaScript SDK를 사용하는 Amazon Cognito 자격 증명 공급자 예제

다음 코드 예제는 Amazon Cognito ID 공급자와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

시작하기

Hello Amazon Cognito

다음 코드 예시에서는 Amazon Cognito 사용을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListUserPools참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

사용자 확인

다음 코드 예제에서는 Amazon Cognito 사용자를 확인하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API ConfirmSignUp참조를 참조하십시오.

추적을 위해 MFA 디바이스 확인

다음 코드 예제에서는 Amazon Cognito에서 추적을 위해 MFA 디바이스를 확인하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API ConfirmDevice참조를 참조하십시오.

MFA 애플리케이션을 사용자와 연결할 수 있는 토큰 가져오기

다음 코드 예제에서는 MFA 애플리케이션을 Amazon Cognito 사용자와 연결할 수 있는 토큰을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API AssociateSoftwareToken참조를 참조하십시오.

사용자 정보 가져오기

다음 코드 예제에서는 Amazon Cognito 사용자에 대한 정보를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
    Username: username,
  });
```

```
  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API AdminGetUser참조를 참조하십시오.

사용자 나열

다음 코드 예제에서는 Amazon Cognito 사용자를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListUsers참조를 참조하십시오.

확인 코드 다시 보내기

다음 코드 예제에서는 Amazon Cognito 확인 코드를 다시 보내는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API ResendConfirmationCode참조를 참조하십시오.

SRP 인증 문제에 응답

다음 코드 예제에서는 Amazon Cognito SRP 인증 문제에 응답하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API RespondToAuthChallenge참조를 참조하십시오.

인증 문제에 응답

다음 코드 예제에서는 Amazon Cognito 인증 문제에 응답하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API AdminRespondToAuthChallenge참조를 참조하십시오.

사용자 가입시키기

다음 코드 예제에서는 사용자를 Amazon Cognito에 가입시키는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
```

```
};
```

- API 세부 정보는 AWS SDK for JavaScript API SignUp참조를 참조하십시오.

인증 시작

다음 코드 예시에서는 Amazon Cognito로 인증을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API InitiateAuth참조를 참조하십시오.

관리자 보안 인증 정보로 인증 시작

다음 코드 예제에서는 Amazon Cognito 및 관리자 보안 인증 정보로 인증을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API AdminInitiateAuth참조를 참조하십시오.

사용자로 MFA 애플리케이션 확인

다음 코드 예제에서는 Amazon Cognito 사용자로 MFA 애플리케이션을 확인하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API VerifySoftwareToken참조를 참조하십시오.

시나리오

MFA가 필요한 사용자 풀에 사용자 가입시키기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 사용자 이름, 암호 및 이메일 주소로 사용자를 가입시키고 확인합니다.
- MFA 애플리케이션을 사용자와 연결하여 다중 인증을 설정합니다.
- 암호와 MFA 코드를 사용하여 로그인합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

최상의 경험을 위해 GitHub 리포지토리를 복제하고 이 예제를 실행하세요. 다음 코드는 전체 예시 애플리케이션의 샘플을 나타냅니다.

```
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 "qrcode-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utils/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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

(v3) 에 JavaScript SDK를 사용하는 DynamoDB 예제

다음 코드 예제는 DynamoDB와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

시작하기

Hello DynamoDB

다음 코드 예제에서는 DynamoDB를 사용하여 시작하는 방법을 보여 줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListTables참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

테이블 생성

다음 코드 예제에서는 DynamoDB 테이블을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateTable참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateTable참조를 참조하십시오.

테이블 삭제

다음 코드 예제에서는 DynamoDB 테이블을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteTable참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteTable참조를 참조하십시오.

테이블에서 항목 삭제

다음 코드 예제에서는 DynamoDB 테이블에서 항목을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API 세부 정보는 을 참조하십시오 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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteItem참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블에서 항목을 삭제합니다.

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

DynamoDB 문서 클라이언트를 사용하여 테이블에서 항목을 삭제합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteItem참조를 참조하십시오.

항목 배치 가져오기

다음 코드 예제에서는 DynamoDB 항목 배치를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API 세부 정보는 을 참조하십시오 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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API BatchGetItem참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API BatchGetItem참조를 참조하십시오.

테이블에서 항목 가져오기

다음 코드 예제는 DynamoDB 테이블에서 항목을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API 세부 정보는 을 참조하십시오 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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetItem참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블에서 항목을 가져옵니다.

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

DynamoDB 문서 클라이언트를 사용하여 테이블에서 항목을 가져옵니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetItem참조를 참조하십시오.

테이블에 대한 정보 가져오기

다음 코드 예제는 DynamoDB 테이블에 대한 정보를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DescribeTable참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DescribeTable참조를 참조하십시오.

테이블 나열

다음 코드 예제에서는 DynamoDB 테이블을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListTables참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListTables참조를 참조하십시오.

테이블에 항목 추가

다음 코드 예제에서는 DynamoDB 테이블에 항목을 추가하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API 세부 정보는 을 참조하십시오 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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API PutItem참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블에 항목을 추가합니다.

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

DynamoDB 문서 클라이언트를 사용하여 테이블에 항목을 추가합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutItem참조를 참조하십시오.

테이블 쿼리

다음 코드 예제에서는 DynamoDB 테이블을 쿼리하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API 세부 정보는 을 참조하십시오 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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하십시오.

PartiQL 문 실행

다음 코드 예제에서는 DynamoDB 테이블에서 PartiQL 문을 실행하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

PartiQL을 사용하여 항목을 생성합니다.

```
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;
};
```

PartiQL을 사용하여 항목을 가져옵니다.

```
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;
};
```

PartiQL을 사용하여 항목을 업데이트합니다.

```
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;
};
```

PartiQL을 사용하여 항목을 삭제합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ExecuteStatement참조를 참조하십시오.

PartiQL 문 배치 실행

다음 코드 예제에서는 DynamoDB 테이블에서 PartiQL 문 배치를 실행하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

PartiQL을 사용하여 항목 배치를 생성합니다.

```
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;
};
```

PartiQL을 사용하여 항목 배치를 가져옵니다.

```
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;
};
```

PartiQL을 사용하여 항목 배치를 업데이트합니다.

```
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;
};
```

PartiQL을 사용하여 항목 배치를 삭제합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API BatchExecuteStatement참조를 참조하십시오.

테이블 스캔

다음 코드 예제에서는 DynamoDB 테이블을 스캔하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API 세부 정보는 을 참조하십시오 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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Scan을 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Scan을 참조하십시오.

테이블의 항목 업데이트

다음 코드 예제에서는 DynamoDB 테이블의 항목을 업데이트하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API 세부 정보는 을 참조하십시오 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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API UpdateItem참조를 참조하십시오.

항목 배치 쓰기

다음 코드 예제에서는 DynamoDB 항목 배치를 쓰는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API 세부 정보는 을 참조하십시오 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);
  }
```

```
};
```

- API 세부 정보는 AWS SDK for JavaScript API BatchWriteItem참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API BatchWriteItem참조를 참조하십시오.

시나리오

테이블, 항목 및 쿼리 시작

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 영화 데이터를 저장할 수 있는 테이블을 생성합니다.
- 테이블에 하나의 영화를 추가하고 가져오고 업데이트합니다.
- 샘플 JSON 파일에서 테이블에 영화 데이터를 씁니다.
- 특정 연도에 개봉된 영화를 쿼리합니다.
- 특정 연도 범위 동안 개봉된 영화를 스캔합니다.
- 테이블에서 영화를 삭제한 다음, 테이블을 삭제합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

PartiQL 문 배치를 사용하여 테이블 쿼리

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 여러 SELECT 문을 실행하여 항목 배치를 가져옵니다.
- 여러 INSERT 문을 실행하여 항목 배치를 추가합니다.
- 여러 UPDATE 문을 실행하여 항목 배치를 업데이트합니다.
- 여러 DELETE 문을 실행하여 항목 배치를 삭제합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

배치 PartiQL 명령문을 실행합니다.

```
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 addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
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.");
};
```

- API 세부 정보는 AWS SDK for JavaScript API BatchExecuteStatement참조를 참조하십시오.

PartiQL을 사용하여 테이블 쿼리

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- SELECT 문을 실행하여 항목을 가져옵니다.
- INSERT 문을 실행하여 항목을 추가합니다.
- UPDATE 문을 실행하여 항목을 업데이트합니다.

- DELETE 문을 실행하여 항목을 삭제합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

단일 PartiQL 문을 실행합니다.

```
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/ql-
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.");
};
```

- API 세부 정보는 AWS SDK for JavaScript API ExecuteStatement참조를 참조하십시오.

(v3) 용 JavaScript SDK를 사용하는 Amazon EC2 예제

다음 코드 예제는 Amazon EC2와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 상황에 GitHub 맞게 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

시작하기

Hello Amazon EC2

다음 코드 예제는 Amazon EC2 사용을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeSecurityGroups참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

탄력적 IP 주소 할당

다음 코드 예제에서는 Amazon EC2에 탄력적 IP 주소를 할당하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API AllocateAddress참조를 참조하십시오.

인스턴스와 탄력적 IP 주소 연결

다음 코드 예제에서는 탄력적 IP 주소를 Amazon EC2 인스턴스와 연결하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API AssociateAddress참조를 참조하십시오.

시작 템플릿 생성

다음 코드 예시에서는 Amazon EC2 시작 템플릿을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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,
    },
  }),
```

- API 세부 정보는 AWS SDK for JavaScript API CreateLaunchTemplate참조를 참조하십시오.

보안 그룹 생성

다음 코드 예제에서는 Amazon EC2 보안 그룹을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateSecurityGroup참조를 참조하십시오.

보안 키 페어 생성

다음 코드 예제에서는 Amazon EC2용 보안 키 페어를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateKeyPair참조를 참조하십시오.

인스턴스 생성 및 실행

다음 코드 예제에서는 Amazon EC2 인스턴스를 생성하고 실행하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API RunInstances참조를 참조하십시오.

시작 템플릿 삭제

다음 코드 예시에서는 Amazon EC2 시작 템플릿을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteLaunchTemplate참조를 참조하십시오.

보안 그룹 삭제

다음 코드 예제에서는 Amazon EC2 보안 그룹을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteSecurityGroup참조를 참조하십시오.

보안 키 페어 삭제

다음 코드 예제에서는 Amazon EC2 보안 키 페어를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteKeyPair참조를 참조하십시오.

리전 설명

다음 코드 예제에서는 Amazon EC2 리전을 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeRegions참조를 참조하십시오.

인스턴스 설명

다음 코드 예제에서는 Amazon EC2 인스턴스를 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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 = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${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);
```

```
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeInstances참조를 참조하십시오.

세부 모니터링 비활성화

다음 코드 예제에서는 Amazon EC2 인스턴스에서 세부 모니터링을 비활성화하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API UnmonitorInstances참조를 참조하십시오.

인스턴스에서 탄력적 IP 주소 연결 해제

다음 코드 예제에서는 Amazon EC2 인스턴스에서 탄력적 IP 주소의 연결을 해제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DisassociateAddress참조를 참조하십시오.

모니터링 활성화

다음 코드 예제에서는 실행 중인 Amazon EC2 인스턴스에 대한 모니터링을 활성화하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API MonitorInstances참조를 참조하십시오.

Amazon Machine Image에 대한 데이터 가져오기

다음 코드 예제에서는 Amazon Machine Image(AMI)에 관한 데이터를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeImages참조를 참조하십시오.

보안 그룹에 대한 데이터 가져오기

다음 코드 예제에서는 Amazon EC2 보안 그룹에 관한 데이터를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeSecurityGroups참조를 참조하십시오.

인스턴스 유형에 대한 데이터 가져오기

다음 코드 예제에서는 Amazon EC2 인스턴스 유형에 관한 데이터를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeInstanceTypes참조를 참조하십시오.

인스턴스와 연결된 인스턴스 프로파일에 대한 데이터 가져오기

다음 코드 예시는 Amazon EC2 인스턴스와 연결된 인스턴스 프로파일에 대한 데이터를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    const ec2Client = new EC2Client({});
    const { IamInstanceProfileAssociations } = await ec2Client.send(
      new DescribeIamInstanceProfileAssociationsCommand({
        Filters: [
          { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
        ],
      }),
    );
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeIamInstanceProfileAssociations참조를 참조하십시오.

탄력적 IP 주소에 대한 세부 정보 가져오기

다음 코드 예제에서는 탄력적 IP 주소에 관한 세부 정보를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeAddresses참조를 참조하십시오.

기본 VPC 가져오기

다음 코드 예시는 현재 계정의 기본 VPC를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeVpcs참조를 참조하십시오.

VPC의 기본 서브넷 가져오기

다음 코드 예시는 VPC의 기본 서브넷을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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"] },
    ],
  }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeSubnets참조를 참조하십시오.

보안 키 페어 나열

다음 코드 예제에서는 Amazon EC2 보안 키 페어를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeKeyPairs참조를 참조하십시오.

인스턴스 재부팅

다음 코드 예제에서는 Amazon EC2 인스턴스를 재부팅하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API RebootInstances참조를 참조하십시오.

탄력적 IP 주소 릴리스

다음 코드 예제에서는 탄력적 IP 주소를 릴리스하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API ReleaseAddress참조를 참조하십시오.

인스턴스와 연결된 인스턴스 프로파일 교체하기

다음 코드 예시는 Amazon EC2 인스턴스와 연결된 인스턴스 프로파일을 교체하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
```

```
    );
```

- API 세부 정보는 AWS SDK for JavaScript API ReplaceIamInstanceProfileAssociation참조를 참조하십시오.

보안 그룹에 대한 인바운드 규칙 설정

다음 코드 예제에서는 Amazon EC2 보안 그룹에 인바운드 규칙을 설정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API AuthorizeSecurityGroupIngress참조를 참조하십시오.

인스턴스 시작

다음 코드 예제에서는 Amazon EC2 인스턴스를 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API StartInstances참조를 참조하십시오.

인스턴스 중지

다음 코드 예제에서는 Amazon EC2 인스턴스를 중지하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
```

```
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API StopInstances참조를 참조하십시오.

인스턴스 종료

다음 코드 예제에서는 Amazon EC2 인스턴스를 종료하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API TerminateInstances참조를 참조하십시오.

시나리오

복원력이 뛰어난 서비스 구축 및 관리

다음 코드 예제에서는 책, 영화, 노래 추천을 반환하는 로드 밸런싱 웹 서비스를 만드는 방법을 보여줍니다. 이 예제에서는 서비스가 장애에 대응하는 방법과 장애 발생 시 복원력을 높이기 위해 서비스를 재구성하는 방법을 보여줍니다.

- Amazon EC2 Auto Scaling 그룹을 사용하여 시작 템플릿을 기반으로 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스를 생성하고 인스턴스 수를 지정된 범위 내로 유지합니다.
- Elastic Load Balancing으로 HTTP 요청을 처리하고 배포합니다.
- 오토 스케일링의 인스턴스 상태를 모니터링하고 요청을 정상 인스턴스로만 전달합니다.
- 각 EC2 인스턴스에서 Python 웹 서버를 실행하여 HTTP 요청을 처리합니다. 웹 서버는 추천 및 상태 확인으로 응답합니다.
- Amazon DynamoDB 테이블을 사용하여 추천 서비스를 시뮬레이션합니다.
- AWS Systems Manager 매개변수를 업데이트하여 요청 및 상태 확인에 대한 웹 서버 응답을 제어합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
#!/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);
}
```

모든 리소스를 배포하기 위한 단계를 생성합니다.

```
// 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,
      );
    }
  }),
];
```

데모를 실행하기 위한 단계를 생성합니다.

```
// 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;
}
```

모든 리소스를 폐기하는 단계를 생성합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer

- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

인스턴스 시작하기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 키 페어 및 보안 그룹을 생성합니다.
- Amazon Machine Image(AMI) 및 호환되는 인스턴스 유형을 선택한 다음 인스턴스를 생성합니다.
- 인스턴스를 중지한 후 다시 시작합니다.
- 인스턴스와 탄력적 IP 주소 연결.
- SSH로 인스턴스에 연결한 다음 리소스를 정리합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
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(
      `\n Instance started. The IP address changed from
${instanceDetails.PublicIpAddress} to ${ipAddressAfterRestart}`,
      `\n${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.`,
      `\n${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.",
    );
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - AllocateAddress
  - AssociateAddress
  - AuthorizeSecurityGroupIngress
  - CreateKeyPair
  - CreateSecurityGroup
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DescribeImages
  - DescribeInstanceTypes
  - DescribeInstances
  - DescribeKeyPairs
  - DescribeSecurityGroups
  - DisassociateAddress
  - ReleaseAddress
  - RunInstances
  - StartInstances
  - StopInstances
  - TerminateInstances

- UnmonitorInstances

JavaScript (v3) 용 SDK를 사용한 Elastic Load Balancing 예제

다음 코드 예제는 Elastic Load Balancing과 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

시작하기

Elastic Load Balancing 시작

다음 코드 예제에서는 Elastic Load Balancing을 사용하여 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API DescribeLoadBalancers참조를 참조하십시오.

주제

작업

로드 밸런서에 대한 리스너 생성

다음 코드 예제에서는 ELB 로드 밸런서의 요청을 대상 그룹에 전달하는 리스너를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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 },
    ],
  }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API CreateListener참조를 참조하십시오.

대상 그룹 생성

다음 코드 예제에서는 ELB 대상 그룹을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API CreateTargetGroup참조를 참조하십시오.

Application Load Balancer 생성

다음 코드 예제에서는 ELB Application Load Balancer를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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] },
);
```

- API 세부 정보는 AWS SDK for JavaScript API CreateLoadBalancer참조를 참조하십시오.

로드 밸런서 삭제

다음 코드 예제에서는 ELB 로스 밸런서를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API DeleteLoadBalancer참조를 참조하십시오.

대상 그룹 삭제

다음 코드 예제에서는 ELB 대상 그룹을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteTargetGroup참조를 참조하십시오.

대상 그룹 설명

다음 코드 예시에서는 특정 대상 그룹을 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeTargetGroups참조를 참조하십시오.

로드 밸런서의 엔드포인트 가져오기

다음 코드 예제에서는 ELB 로드 밸런서의 엔드포인트를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API DescribeLoadBalancers참조를 참조하십시오.

대상 그룹의 상태 가져오기

다음 코드 예제에서는 ELB 대상 그룹에 있는 인스턴스의 상태를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeTargetHealth참조를 참조하십시오.

시나리오

복원력이 뛰어난 서비스 구축 및 관리

다음 코드 예제에서는 책, 영화, 노래 추천을 반환하는 로드 밸런싱 웹 서비스를 만드는 방법을 보여줍니다. 이 예제에서는 서비스가 장애에 대응하는 방법과 장애 발생 시 복원력을 높이기 위해 서비스를 재구성하는 방법을 보여줍니다.

- Amazon EC2 Auto Scaling 그룹을 사용하여 시작 템플릿을 기반으로 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스를 생성하고 인스턴스 수를 지정된 범위 내로 유지합니다.
- Elastic Load Balancing으로 HTTP 요청을 처리하고 배포합니다.
- 오토 스케일링의 인스턴스 상태를 모니터링하고 요청을 정상 인스턴스로만 전달합니다.
- 각 EC2 인스턴스에서 Python 웹 서버를 실행하여 HTTP 요청을 처리합니다. 웹 서버는 추천 및 상태 확인으로 응답합니다.
- Amazon DynamoDB 테이블을 사용하여 추천 서비스를 시뮬레이션합니다.
- AWS Systems Manager 매개변수를 업데이트하여 요청 및 상태 확인에 대한 웹 서버 응답을 제어합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
#!/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);
}
```

모든 리소스를 배포하기 위한 단계를 생성합니다.

```
// 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,
      );
    }
  }),
];
```

데모를 실행하기 위한 단계를 생성합니다.

```
// 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;
}
```

모든 리소스를 폐기하는 단계를 생성합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances

- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

EventBridge JavaScript (v3) 용 SDK 사용 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. EventBridge

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업

작업

대상 추가

다음 코드 예제는 Amazon EventBridge 이벤트에 대상을 추가하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API PutTargets참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API PutTargets참조를 참조하십시오.

규칙 생성

다음 코드 예제는 Amazon EventBridge 규칙을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
```

```
};
```

- API 세부 정보는 AWS SDK for JavaScript API PutRule참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API PutRule참조를 참조하십시오.

이벤트 전송

다음 코드 예제는 Amazon EventBridge 이벤트를 전송하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API PutEvents참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API PutEvents참조를 참조하십시오.

AWS Glue JavaScript (v3) 용 SDK 사용 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. AWS Glue

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

시작하기

안녕하세요. AWS Glue

다음 코드 예제에서는 AWS Glue의 사용을 시작하는 방법을 보여 줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListJobs참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

크롤러 만들기

다음 코드 예제는 AWS Glue 크롤러를 만드는 방법을 보여줍니다.

(v3) 용 JavaScript SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateCrawler참조를 참조하십시오.

작업 정의 생성

다음 코드 예제는 AWS Glue 작업 정의를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateJob참조를 참조하십시오.

크롤러 삭제

다음 코드 예제는 AWS Glue 크롤러를 삭제하는 방법을 보여줍니다.

(v3) 용 JavaScript SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteCrawler참조를 참조하십시오.

데이터 카탈로그에서 데이터베이스 삭제

다음 코드 예제에서는 AWS Glue Data Catalog에서 데이터베이스를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const deleteDatabase = (databaseName) => {
```

```
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteDatabase참조를 참조하십시오.

작업 정의 삭제

다음 코드 예제는 AWS Glue 작업 정의 및 모든 관련 실행을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteJob참조를 참조하십시오.

데이터베이스에서 테이블 삭제

다음 코드 예제는 AWS Glue Data Catalog 데이터베이스에서 테이블을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteTable참조를 참조하십시오.

크롤러 가져오기

다음 코드 예제는 AWS Glue 크롤러를 가져오는 방법을 보여줍니다.

(v3) 용 JavaScript SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
```

```
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetCrawler참조를 참조하십시오.

데이터 카탈로그에서 데이터베이스 가져오기

다음 코드 예제에서는 AWS Glue Data Catalog에서 데이터베이스를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetDatabase참조를 참조하십시오.

작업 실행 가져오기

다음 코드 예제는 AWS Glue 작업을 실행하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetJobRun참조를 참조하십시오.

데이터 카탈로그에서 데이터베이스 가져오기

다음 코드 예제에서는 AWS Glue Data Catalog에서 데이터베이스 목록을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
```

```
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetDatabases참조를 참조하십시오.

데이터 카탈로그에서 작업 가져오기

다음 코드 예제에서는 AWS Glue Data Catalog에서 작업을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetJob참조를 참조하십시오.

작업 실행 가져오기

다음 코드 예제는 AWS Glue 작업을 실행하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetJobRuns참조를 참조하십시오.

데이터베이스에서 테이블 가져오기

다음 코드 예제에서는 AWS Glue Data Catalog의 데이터베이스에서 테이블을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetTables참조를 참조하십시오.

작업 정의 나열

다음 코드 예제는 AWS Glue 작업 정의를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListJobs참조를 참조하십시오.

크롤러 시작

다음 코드 예제는 AWS Glue 크롤러를 시작하는 방법을 보여줍니다.

(v3) 용 JavaScript SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
```

```
    Name: name,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API StartCrawler참조를 참조하십시오.

작업 실행 시작

다음 코드 예제는 AWS Glue 작업 실행을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API StartJobRun참조를 참조하십시오.

시나리오

크롤러 및 작업 시작하기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 퍼블릭 Amazon S3 버킷을 크롤링하고 CSV 형식의 메타데이터 데이터베이스를 생성하는 크롤러를 생성합니다.
- 에 있는 데이터베이스 및 테이블에 대한 정보를 나열하십시오 AWS Glue Data Catalog.
- 작업을 생성하여 S3 버킷에서 CSV 데이터를 추출하고, 데이터를 변환하며, JSON 형식의 출력을 다른 S3 버킷으로 로드합니다.
- 작업 실행에 대한 정보를 나열하고 변환된 데이터를 확인하며 리소스를 정리합니다.

자세한 내용은 자습서: AWS Glue Studio 시작하기를 참조하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

공용 Amazon Simple Storage Service (S3) 버킷을 크롤링하고 검색한 CSV 형식의 데이터를 설명하는 메타데이터 데이터베이스를 생성하는 크롤러를 만들고 실행합니다.

```
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 };
  };
```

내 데이터베이스 및 테이블에 대한 정보를 나열하세요 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 };
  };
```

소스 Amazon S3 버킷에서 CSV 데이터를 추출하고, 필드를 제거하고 이름을 변경하여 변환하고, JSON 형식의 출력을 다른 Amazon S3 버킷으로 로드하는 작업을 만들고 실행합니다.

```
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 };
  };
```

작업 실행에 대한 정보를 나열하고 변환된 데이터 중 일부를 볼 수 있습니다.

```
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 };
  };
```

데모 중에 생성된 모든 리소스를 삭제합니다.

```
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 };
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

HealthImaging JavaScript (v3) 용 SDK 사용 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. HealthImaging

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

시작하기

안녕하세요. HealthImaging

다음 코드 예제에서는 HealthImaging의 사용을 시작하는 방법을 보여 줍니다.

JavaScript (v3) 용 SDK

```
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;
};
```

- API에 대한 자세한 내용은 API ListDatastores레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

주제

작업

리소스에 태그를 추가

다음 코드 예제는 HealthImaging 리소스에 태그를 추가하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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;
};
```

- API에 대한 자세한 내용은 API TagResource레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트 복사

다음 코드 예제는 HealthImaging 이미지 세트를 복사하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

이미지 세트를 복사하는 유틸리티 함수입니다.

```
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 = "xxxxxxxxxxx",
```

```
  imageSetId = "xxxxxxxxxxx",
  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: 'xxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             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:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

대상 없이 이미지 세트를 복사합니다.

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

대상이 있는 이미지 세트를 복사합니다.

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "4",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

- API에 대한 자세한 내용은 API CopyImageSet레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

데이터 스토어 생성

다음 코드 예제는 HealthImaging 데이터 저장소를 만드는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //    datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- API에 대한 자세한 내용은 API CreateDatastore레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

데이터 스토어 삭제

다음 코드 예제는 HealthImaging 데이터 저장소를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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;
};
```

- API에 대한 자세한 내용은 API DeleteDatastore레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트 삭제

다음 코드 예제는 HealthImaging 이미지 세트를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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 = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
```

```
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- API에 대한 자세한 내용은 API DeleteImageSet레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 프레임 가져오기

다음 코드 예제에서는 이미지 프레임을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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;
};
```

- API에 대한 자세한 내용은 API GetImageFrame레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

데이터 스토어 속성 가져오기

다음 코드 예제는 HealthImaging 데이터 저장소 속성을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response["datastoreProperties"];
};
```

- API에 대한 자세한 내용은 API GetDatastore레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트 속성 가져오기

다음 코드 예제는 HealthImaging 이미지 세트 속성을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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 = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  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
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }
```

```
  return response;
};
```

- API에 대한 자세한 내용은 API GetImageSet레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

가져오기 작업 속성 가져오기

다음 코드 예제에서는 가져오기 작업 속성을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 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: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- API에 대한 자세한 내용은 API 레퍼런스의 GetDiCom을 ImportJob 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트의 메타데이터 가져오기

다음 코드 예제는 HealthImaging 이미지 세트의 메타데이터를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

이미지 세트 메타데이터를 가져오는 유틸리티 함수입니다.

```
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 = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  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;
};
```

버전 없이 이미지 세트 메타데이터를 가져옵니다.

```
  try {
```

```
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

버전과 함께 이미지 세트 메타데이터를 가져옵니다.

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- API에 대한 자세한 내용은 API GetImageSetMetadata레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

데이터 스토어로 대량 데이터 가져오기

다음 코드 예제는 대량 데이터를 HealthImaging 데이터 저장소로 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- API에 대한 자세한 내용은 API 레퍼런스의 ImportJob 스타트디컴을 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

데이터 스토어 목록

다음 코드 예제는 HealthImaging 데이터 저장소를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
```

```
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- API에 대한 자세한 내용은 API ListDatastores레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트 버전 나열

다음 코드 예제는 HealthImaging 이미지 세트 버전을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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 = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx"
) => {
  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: 'xxxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- API에 대한 자세한 내용은 API ListImageSetVersions레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

데이터 스토어의 가져오기 작업 나열

다음 코드 예제는 HealthImaging 데이터 저장소의 가져오기 작업을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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 = "xxxxxxxxxxxxxxxxx"
) => {
  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: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- API에 대한 자세한 내용은 API 레퍼런스의 ListDicom을 ImportJobs 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

리소스에 대한 태그를 나열합니다.

다음 코드 예제는 HealthImaging 리소스의 태그를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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;
};
```

- API에 대한 자세한 내용은 API ListTagsForResource레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

리소스에서 태그를 제거

다음 코드 예제는 HealthImaging 리소스에서 태그를 제거하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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;
};
```

- API에 대한 자세한 내용은 API UntagResource레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트 검색

다음 코드 예제는 HealthImaging 이미지 세트를 검색하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

이미지 세트 검색을 위한 유틸리티 함수입니다.

```
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;
};
```

사용 사례 #1: EQUAL 연산자.

```
  const datastoreId = "12345678901234567890123456789012";

  try {
    const filters = [
      {
        values: [{ DICOMPatientId: "9227465" }],
        operator: "EQUAL",
      },
    ];

    await searchImageSets(datastoreId, filters);
  } catch (err) {
    console.error(err);
  }
```

사용 사례 #2: DICOM과 DICOM을 StudyDate 사용하는 사업자 간. StudyTime

```
  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);
}
```

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
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);
  }
```

- API에 대한 자세한 내용은 API SearchImageSets레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트 메타데이터 업데이트

다음 코드 예제는 HealthImaging 이미지 세트 메타데이터를 업데이트하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

```
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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
};
```

메타데이터를 인코딩합니다.

```
    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);
```

- API에 대한 자세한 내용은 API UpdateImageSetMetadata레퍼런스를 참조하십시오.AWS SDK for JavaScript

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

시나리오

데이터 저장소에 태그 지정

다음 코드 예제는 HealthImaging 데이터 저장소에 태그를 지정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

데이터 스토어에 태깅하려면.

```
  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);
  }
```

리소스에 태그를 지정하는 유틸리티 함수.

```
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;
};
```

데이터 스토어의 태그를 나열하려면.

```
  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);
  }
```

리소스의 태그를 나열하는 유틸리티 함수입니다.

```
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;
};
```

데이터 스토어에 태그 지정을 해제하려면.

```
  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);
  }
```

리소스의 태그를 해제하는 유틸리티 함수.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트 태그 지정

다음 코드 예제는 HealthImaging 이미지 세트에 태그를 지정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

이미지 세트에 태그를 지정하려면.

```
  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);
  }
```

리소스에 태그를 지정하는 유틸리티 함수.

```
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;
};
```

이미지 세트의 태그를 나열하려면.

```
  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);
  }
```

리소스의 태그를 나열하는 유틸리티 함수입니다.

```
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;
};
```

이미지 세트의 태그를 해제하려면.

```
  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);
  }
```

리소스의 태그를 해제하는 유틸리티 함수.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript (v3) 용 SDK를 사용한 IAM 예제

다음 코드 예제는 IAM과 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

시작하기

Hello IAM

다음 코드 예제에서는 IAM을 사용하여 시작하는 방법을 보여 줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API ListPolicies참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

역할에 정책 연결

다음 코드 예제에서는 IAM 정책을 역할에 연결하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

정책을 연결합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API AttachRolePolicy참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API AttachRolePolicy참조를 참조하십시오.

역할에 인라인 정책 연결

다음 코드 예제에서는 인라인 정책을 IAM 역할에 연결하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API PutRolePolicy참조를 참조하십시오.

SAML 공급자를 생성합니다.

다음 코드 예제는 AWS Identity and Access Management (IAM) SAML 공급자를 생성하는 방법을 보여줍니다.

(v3) 용 JavaScript SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 CreateSAMLProvider를 참조하십시오.

그룹 생성

다음 코드 예제에서는 IAM 그룹을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateGroup참조를 참조하십시오.

정책 생성

다음 코드 예제에서는 IAM 정책을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

정책을 생성합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreatePolicy참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreatePolicy참조를 참조하십시오.

역할 생성

다음 코드 예제에서는 IAM 역할을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

역할을 생성합니다.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateRole참조를 참조하십시오.

서비스 연결 역할 생성

다음 코드 예제에서는 IAM 서비스 연결 역할을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

서비스 연결 역할을 생성합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateServiceLinkedRole참조를 참조하십시오.

사용자 생성

다음 코드 예제에서는 IAM 사용자를 생성하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

사용자를 생성합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateUser참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateUser참조를 참조하십시오.

액세스 키 생성

다음 코드 예제에서는 IAM 액세스 키를 생성하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

액세스 키를 생성합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateAccessKey참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.

- API 세부 정보는 AWS SDK for JavaScript API CreateAccessKey참조를 참조하십시오.

계정의 별칭 생성

다음 코드 예제에서는 IAM 계정의 별칭을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

계정 별칭을 생성합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateAccountAlias참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateAccountAlias참조를 참조하십시오.

인스턴스 프로파일 생성

다음 코드 예제에서는 IAM 인스턴스 프로파일을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
```

- API 세부 정보는 AWS SDK for JavaScript API CreateInstanceProfile참조를 참조하십시오.

SAML 공급자 삭제

다음 코드 예제는 AWS Identity and Access Management (IAM) SAML 공급자를 삭제하는 방법을 보여줍니다.

(v3) 용 JavaScript SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteSAMLProvider를 참조하십시오.

그룹 삭제

다음 코드 예제에서는 IAM 그룹을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteGroup참조를 참조하십시오.

정책 삭제

다음 코드 예제에서는 IAM 정책을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

정책을 삭제합니다.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeletePolicy참조를 참조하십시오.

역할 삭제

다음 코드 예제에서는 IAM 역할을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

역할을 삭제합니다.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteRole참조를 참조하십시오.

역할 정책 제거

다음 코드 예제에서는 IAM 역할 정책을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteRolePolicy참조를 참조하십시오.

서버 인증서 삭제

다음 코드 예제에서는 IAM 서버 인증서를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

서버 인증서를 삭제합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.

- API 세부 정보는 AWS SDK for JavaScript API DeleteServerCertificate참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteServerCertificate참조를 참조하십시오.

서비스 연결 역할 삭제

다음 코드 예제에서는 IAM 서비스 연결 역할을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteServiceLinkedRole참조를 참조하십시오.

사용자 삭제

다음 코드 예제에서는 IAM 사용자를 삭제하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

사용자를 삭제합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteUser참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteUser참조를 참조하십시오.

액세스 키 삭제

다음 코드 예제에서는 IAM 액세스 키를 삭제하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

액세스 키를 삭제합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteAccessKey참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteAccessKey참조를 참조하십시오.

계정 별칭 삭제

다음 코드 예제에서는 IAM 계정 별칭을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

계정 별칭을 삭제합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteAccountAlias참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteAccountAlias참조를 참조하십시오.

인스턴스 프로파일 삭제

다음 코드 예제에서는 IAM 인스턴스 프로파일을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const client = new IAMClient({});
await client.send(
  new DeleteInstanceProfileCommand({
    InstanceProfileName: NAMES.instanceProfileName,
  }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteInstanceProfile참조를 참조하십시오.

역할에서 정책 분리

다음 코드 예제에서는 역할에서 IAM 정책을 분리하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

태그를 분리합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DetachRolePolicy참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DetachRolePolicy참조를 참조하십시오.

정책 가져오기

다음 코드 예제에서는 IAM 정책을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

정책을 가져옵니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetPolicy참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetPolicy참조를 참조하십시오.

역할 가져오기

다음 코드 예제에서는 IAM 역할을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

역할을 가져옵니다.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetRole참조를 참조하십시오.

서버 인증서 조회

다음 코드 예제에서는 IAM 서버 인증서를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

서버 인증서를 가져옵니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetServerCertificate참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetServerCertificate참조를 참조하십시오.

서비스 연결 역할의 삭제 상태 확인

다음 코드 예제는 AWS Identity and Access Management (IAM) 서비스 연결 역할의 삭제 상태를 가져오는 방법을 보여줍니다.

(v3) 용 SDK JavaScript

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetServiceLinkedRoleDeletionStatus참조를 참조하십시오.

액세스 키의 마지막 사용에 대한 데이터 가져오기

다음 코드 예제에서는 IAM 액세스 키의 마지막 사용에 관한 데이터를 가져오는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

액세스 키를 가져옵니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetAccessKeyLastUsed참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetAccessKeyLastUsed참조를 참조하십시오.

계정 암호 정책 가져오기

다음 코드 예제에서는 IAM 계정 암호 정책을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

계정 암호 정책을 가져옵니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetAccountPasswordPolicy참조를 참조하십시오.

SAML 공급자 나열

다음 코드 예제에서는 IAM에 대한 SAML 공급자를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SAML 공급자를 나열합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListSAMLProviders를 참조하십시오.

사용자의 액세스 키 나열

다음 코드 예제에서는 사용자의 IAM 액세스 키를 나열하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

액세스 키를 나열합니다.

```
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;
    }
  }
}
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListAccessKeys참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListAccessKeys참조를 참조하십시오.

계정 별칭 조회

다음 코드 예제에서는 IAM 계정 별칭을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

계정 별칭을 나열합니다.

```
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;
    }
  }
}
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListAccountAliases참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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);
```

```
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListAccountAliases참조를 참조하십시오.

그룹 나열

다음 코드 예제에서는 IAM 그룹을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

그룹을 나열합니다.

```
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;
    }
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API ListGroups참조를 참조하십시오.

역할에 대한 인라인 정책 나열

다음 코드 예제에서는 IAM 역할에 대한 인라인 정책을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

정책을 나열합니다.

```
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;
    }
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API ListRolePolicies참조를 참조하십시오.

정책 나열

다음 코드 예제에서는 IAM 정책을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

정책을 나열합니다.

```
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;
    }
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API ListPolicies참조를 참조하십시오.

역할에 연결된 정책 나열

다음 코드 예제에서는 IAM 역할에 연결된 정책을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

역할에 연결된 정책을 나열합니다.

```
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;
    }
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API ListAttachedRolePolicies참조를 참조하십시오.

역할 목록

다음 코드 예제에서는 IAM 역할을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

역할을 나열합니다.

```
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;
    }
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API ListRoles참조를 참조하십시오.

서버 인증서 나열

다음 코드 예제에서는 IAM 서버 인증서를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

인증서를 나열합니다.

```
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;
    }
  }
}
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListServerCertificates참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListServerCertificates참조를 참조하십시오.

사용자 나열

다음 코드 예제에서는 IAM 사용자를 나열하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

사용자를 나열합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListUsers참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListUsers참조를 참조하십시오.

서버 인증서 업데이트

다음 코드 예제에서는 IAM 서버 인증서를 업데이트하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

서버 인증서를 업데이트합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API UpdateServerCertificate참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API UpdateServerCertificate참조를 참조하십시오.

사용자 업데이트

다음 코드 예제에서는 IAM 사용자를 업데이트하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

사용자를 업데이트합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API UpdateUser참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API UpdateUser참조를 참조하십시오.

액세스 키 업데이트

다음 코드 예제에서는 IAM 액세스 키를 업데이트하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

액세스 키를 업데이트합니다.

```
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);
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API UpdateAccessKey참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API UpdateAccessKey참조를 참조하십시오.

서버 인증서 업로드

다음 코드 예제는 AWS Identity and Access Management (IAM) 서버 인증서를 업로드하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API UploadServerCertificate참조를 참조하십시오.

시나리오

복원력이 뛰어난 서비스 구축 및 관리

다음 코드 예제에서는 책, 영화, 노래 추천을 반환하는 로드 밸런싱 웹 서비스를 만드는 방법을 보여줍니다. 이 예제에서는 서비스가 장애에 대응하는 방법과 장애 발생 시 복원력을 높이기 위해 서비스를 재구성하는 방법을 보여줍니다.

- Amazon EC2 Auto Scaling 그룹을 사용하여 시작 템플릿을 기반으로 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스를 생성하고 인스턴스 수를 지정된 범위 내로 유지합니다.
- Elastic Load Balancing으로 HTTP 요청을 처리하고 배포합니다.
- 오토 스케일링의 인스턴스 상태를 모니터링하고 요청을 정상 인스턴스로만 전달합니다.
- 각 EC2 인스턴스에서 Python 웹 서버를 실행하여 HTTP 요청을 처리합니다. 웹 서버는 추천 및 상태 확인으로 응답합니다.
- Amazon DynamoDB 테이블을 사용하여 추천 서비스를 시뮬레이션합니다.
- AWS Systems Manager 매개변수를 업데이트하여 요청 및 상태 확인에 대한 웹 서버 응답을 제어합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
#!/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);
}
```

모든 리소스를 배포하기 위한 단계를 생성합니다.

```
// 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,
      );
    }
  }),
];
```

데모를 실행하기 위한 단계를 생성합니다.

```
// 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;
}
```

모든 리소스를 폐기하는 단계를 생성합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances

- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

사용자 생성 및 역할 수임

다음 코드 예제에서는 사용자를 생성하고 역할을 수임하는 방법을 보여줍니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마십시오. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

- 권한이 없는 사용자를 생성합니다.
- 계정에 대한 Amazon S3 버킷을 나열할 수 있는 권한을 부여하는 역할을 생성합니다.
- 사용자가 역할을 수임할 수 있도록 정책을 추가합니다.
- 역할을 수임하고 임시 보안 인증 정보를 사용하여 S3 버킷을 나열한 후 리소스를 정리합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon S3 버킷을 나열할 수 있는 권한을 부여하는 역할과 IAM 사용자를 생성합니다. 사용자는 역할을 수임할 수 있는 권한만 있습니다. 역할을 수임한 후 임시 자격 증명을 사용하여 계정의 버킷을 나열합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

(v3) 에 JavaScript SDK를 사용하는 Lambda 예제

다음 코드 예제는 Lambda와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

시작하기

Hello Lambda

다음 코드 예제에서는 Lambda를 사용하여 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListFunctions참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

함수 생성

다음 코드 예제에서는 Lambda 함수를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateFunction참조를 참조하십시오.

함수 삭제

다음 코드 예제에서는 Lambda 함수를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteFunction참조를 참조하십시오.

함수 가져오기

다음 코드 예제에서는 Lambda 함수를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetFunction참조를 참조하십시오.

함수 간접 호출

다음 코드 예제에서는 Lambda 함수를 간접적으로 호출하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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 };
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 간접적 호출을 참조하세요.

함수 나열

다음 코드 예제에서는 Lambda 함수를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});
```

```
  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListFunctions참조를 참조하십시오.

함수 코드 업데이트

다음 코드 예제에서는 Lambda 함수를 업데이트하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API UpdateFunctionCode참조를 참조하십시오.

함수 구성 업데이트

다음 코드 예제에서는 Lambda 함수 구성을 업데이트하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API UpdateFunctionConfiguration참조를 참조하십시오.

시나리오

함수 시작하기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- IAM 역할과 Lambda 함수를 생성하고 핸들러 코드를 업로드합니다.
- 단일 파라미터로 함수를 간접적으로 호출하고 결과를 가져옵니다.
- 함수 코드를 업데이트하고 환경 변수로 구성합니다.
- 새 파라미터로 함수를 간접적으로 호출하고 결과를 가져옵니다. 반환된 실행 로그를 표시합니다.
- 계정의 함수를 나열합니다.

자세한 내용은 콘솔로 Lambda 함수 생성을 참조하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Lambda에 로그에 쓸 수 있는 권한을 부여하는 AWS Identity and Access Management (IAM) 역할을 생성합니다.

```
    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);
};
```

Lambda 함수를 생성하고 핸들러 코드를 업로드합니다.

```
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);
};
```

단일 파라미터로 함수를 간접적으로 호출하고 결과를 가져옵니다.

```
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 };
};
```

함수 코드를 업데이트하고 환경 변수를 사용하여 Lambda 환경을 구성합니다.

```
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);
};
```

계정의 함수를 나열합니다.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

IAM 역할과 Lambda 함수를 삭제합니다.

```
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);
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

(v3) 용 JavaScript SDK를 사용한 Amazon Personalize 예제

다음 코드 예제는 Amazon Personalize와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

- 작업

작업

배치 인터페이스 작업 생성

다음 코드 예제에서는 Amazon Personalize 배치 인터페이스 작업을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateBatchInferenceJob참조를 참조하십시오.

배치 세그먼트 작업 생성

다음 코드 예제에서는 Amazon Personalize 배치 세그먼트 작업을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateBatchSegmentJob참조를 참조하십시오.

캠페인 생성

다음 코드 예제에서는 Amazon Personalize 캠페인을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateCampaign참조를 참조하십시오.

데이터 세트 생성

다음 코드 예제에서는 Amazon Personalize 데이터 세트를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateDataset참조를 참조하십시오.

데이터 집합 내보내기 작업 생성

다음 코드 예제에서는 Amazon Personalize 데이터 세트 내보내기 작업을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateDatasetExportJob참조를 참조하십시오.

데이터세트 그룹을 생성

다음 코드 예제에서는 Amazon Personalize 데이터 세트 그룹을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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);
```

도메인 데이터 세트 그룹을 생성합니다.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateDatasetGroup참조를 참조하십시오.

데이터 세트 가져오기 작업을 생성

다음 코드 예제에서는 Amazon Personalize 데이터 세트 가져오기 작업을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateDatasetImportJob참조를 참조하십시오.

도메인 스키마 생성

다음 코드 예제에서는 Amazon Personalize 도메인 스키마를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateSchema참조를 참조하십시오.

필터를 생성

다음 코드 예제에서는 Amazon Personalize 필터를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateFilter참조를 참조하십시오.

추천자 생성

다음 코드 예제에서는 Amazon Personalize 추천을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateRecommender참조를 참조하십시오.

스키마를 생성

다음 코드 예제에서는 Amazon Personalize 스키마를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateSchema참조를 참조하십시오.

솔루션을 생성

다음 코드 예제에서는 Amazon Personalize 솔루션을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateSolution참조를 참조하십시오.

솔루션 버전을 생성

다음 코드 예제에서는 Amazon Personalize 솔루션을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateSolutionVersion참조를 참조하십시오.

이벤트 추적기 생성

다음 코드 예제에서는 Amazon Personalize 이벤트 추적기를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateEventTracker참조를 참조하십시오.

(v3) 용 JavaScript SDK를 사용한 Amazon Personalize 이벤트 예제

다음 코드 예제는 Amazon Personalize Events와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 상황에 GitHub 맞게 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

작업

데이터 세트로 항목 가져오기

다음 코드 예제에서는 Amazon Personalize 이벤트 데이터 세트로 항목을 점진적으로 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API PutItems참조를 참조하십시오.

실시간 상호 작용 이벤트 데이터 가져오기

다음 코드 예제에서는 Amazon Personalize 이벤트로 실시간 상호 작용 이벤트 데이터를 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API PutEvents참조를 참조하십시오.

사용자를 점진적으로 가져오기

다음 코드 예제에서는 Amazon Personalize Events 이벤트로 사용자를 점진적으로 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API PutUsers참조를 참조하십시오.

(v3) 용 JavaScript SDK를 사용한 Amazon Personalize 런타임 예제

다음 코드 예제는 Amazon Personalize Runtime과 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

- 작업

작업

추천 받기(사용자 지정 데이터세트 그룹)

다음 코드 예제에서는 Amazon Personalize Runtime 런타임 순위 권장 사항을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API GetPersonalizedRanking참조를 참조하십시오.

추천에서 권장 사항(도메인 데이터 세트 그룹) 가져오기

다음 코드 예제에서는 Amazon Personalize Runtime 런타임 권장 사항을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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();
```

필터를 사용하여 권장 사항(사용자 지정 데이터 세트 그룹)을 가져옵니다.

```
// 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();
```

도메인 데이터 세트 그룹에 생성된 추천에서 필터링된 권장 사항을 가져옵니다.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API GetRecommendations참조를 참조하십시오.

(v3) 용 JavaScript SDK를 사용한 Amazon Pinpoint 예제

다음 코드 예제는 Amazon Pinpoint와 함께 AWS SDK for JavaScript (v3) 를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

- 작업

작업

이메일 및 문자 메시지 전송

다음 코드 예제에서는 Amazon Pinpoint를 사용하여 이메일 및 문자 메시지를 전송하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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 };
```

이메일 메시지를 전송합니다.

```
// 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 = `<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 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();
```

SMS 메시지를 전송합니다.

```
// 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., XXXXXXXX66e4e9986478cXXXXXXXXX

/* 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();
```

- API 세부 정보는 AWS SDK for JavaScript API SendMessages참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이메일 메시지를 전송합니다.

```
"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 = `<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.sendMessages(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"]
    );
  }
});
```

SMS 메시지를 전송합니다.

```
"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.sendMessages(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"]
    );
  }
});
```

- API 세부 정보는 AWS SDK for JavaScript API SendMessages참조를 참조하십시오.

(v3) 용 JavaScript SDK를 사용하는 Amazon Redshift 예제

다음 코드 예제는 Amazon Redshift와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

- 작업

작업

클러스터 생성

다음 코드 예제에서는 Amazon Redshift 클러스터를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

클러스터를 생성합니다.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API CreateCluster참조를 참조하십시오.

클러스터 삭제

다음 코드 예제에서는 Amazon Redshift 클러스터를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

클러스터를 생성합니다.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteCluster참조를 참조하십시오.

클러스터 설명

다음 코드 예제에서는 Amazon Redshift 클러스터를 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

클러스터에 대해 설명합니다.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeClusters참조를 참조하십시오.

클러스터 수정

다음 코드 예제에서는 Amazon Redshift 클러스터를 수정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

클러스터를 수정합니다.

```
// 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();
```

- API 세부 정보는 AWS SDK for JavaScript API ModifyCluster참조를 참조하십시오.

JavaScript (v3) 용 SDK를 사용하는 Amazon S3 예제

다음 코드 예제는 Amazon S3와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

시작하기

Hello Amazon S3

다음 코드 예시에서는 Amazon S3를 사용하여 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListBuckets참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

버킷에 CORS 규칙 추가

다음 코드 예제에서는 S3 버킷에 교차 오리진 리소스 공유(CORS) 규칙을 추가하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

CORS 규칙을 추가합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutBucketCors참조를 참조하십시오.

버킷에 정책 추가

다음 코드 예제에서는 S3 버킷에 정책을 추가하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

정책을 추가합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutBucketPolicy참조를 참조하십시오.

버킷 간 객체 복사

다음 코드 예제에서는 버킷 간 S3 객체를 복사하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

객체를 복사합니다.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API CopyObject참조를 참조하십시오.

버킷 생성

다음 코드 예제에서는 S3 버킷 생성 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷을 생성합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateBucket참조를 참조하십시오.

버킷에서 정책 삭제

다음 코드 예제에서는 S3 버킷에서 정책을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷 정책을 삭제합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteBucketPolicy참조를 참조하십시오.

빈 버킷 삭제

다음 코드 예제에서는 빈 S3 버킷 삭제 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷을 삭제합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteBucket참조를 참조하십시오.

객체 삭제

다음 코드 예제에서는 S3 객체 삭제 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

객체를 삭제합니다.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteObject참조를 참조하십시오.

여러 객체 삭제

다음 코드 예제에서는 S3 버킷에서 여러 객체를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

여러 객체를 삭제합니다.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteObjects참조를 참조하십시오.

버킷에서 웹 사이트 구성 삭제

다음 코드 예제에서는 S3 버킷에서 웹 사이트 구성을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷에서 웹 사이트 구성을 삭제합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteBucketWebsite참조를 참조하십시오.

버킷에 대한 CORS 규칙 가져오기

다음 코드 예제에서는 S3 버킷에 대한 교차 오리진 리소스 공유(CORS) 규칙을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷에 대한 CORS 정책을 가져옵니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetBucketCors참조를 참조하십시오.

버킷에서 객체 가져오기

다음 코드 예제에서는 S3 버킷의 객체에서 데이터를 읽는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

객체를 다운로드합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetObject참조를 참조하십시오.

버킷의 ACL 가져오기

다음 코드 예제에서는 S3 버킷의 액세스 제어 목록(ACL)을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

ACL 권한을 가져옵니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetBucketAcl참조를 참조하십시오.

버킷에 대한 정책 가져오기

다음 코드 예제에서는 S3 버킷에 대한 정책을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷 정책을 가져옵니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetBucketPolicy참조를 참조하십시오.

버킷에 대한 웹 사이트 구성 가져오기

다음 코드 예제에서는 S3 버킷에 대한 웹 사이트 구성을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

웹 사이트 구성을 가져옵니다.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetBucketWebsite참조를 참조하십시오.

버킷 나열

다음 코드 예제에서는 S3 버킷을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷을 나열합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListBuckets참조를 참조하십시오.

버킷의 객체 나열

다음 코드 예제에서는 S3 버킷의 객체를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷의 모든 객체를 나열합니다. 객체가 두 개 이상 있는 경우, IsTruncated 전체 목록을 반복하는 데 사용됩니다. NextContinuationToken

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListObjects V2를 참조하십시오.

버킷에 대해 새 ACL 설정

다음 코드 예제에서는 S3 버킷의 새 액세스 제어 목록(ACL)을 설정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷 ACL을 적용합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutBucketAcl참조를 참조하십시오.

버킷에 대한 웹 사이트 구성 설정

다음 코드 예제에서는 S3 버킷에 대한 웹 사이트 구성을 설정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

웹 사이트 구성을 설정합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutBucketWebsite참조를 참조하십시오.

버킷에 객체 업로드

다음 코드 예제에서는 S3 버킷에 객체를 업로드하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

객체를 업로드합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API PutObject참조를 참조하십시오.

시나리오

미리 서명된 URL 생성

다음 코드 예제에서는 Amazon S3에 대해 미리 서명된 URL을 생성하고 객체를 업로드하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

미리 서명된 URL을 생성하여 버킷에 객체를 업로드합니다.

```
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);
  }
};
```

미리 서명된 URL을 생성하여 버킷에서 객체를 다운로드합니다.

```
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);
  }
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.

Amazon S3 객체를 나열하는 웹 페이지 생성

다음 코드 예시는 웹 페이지에 Amazon S3 객체를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

다음 코드는 AWS SDK를 호출하는 관련 React 컴포넌트입니다. 이 컴포넌트를 포함하는 실행 가능한 애플리케이션 버전은 이전 링크에서 찾을 수 있습니다. 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;
```

- API 세부 정보는 API 참조를 참조하십시오 ListObjects.AWS SDK for JavaScript

버킷 및 객체 시작하기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 버킷을 만들고 버킷에 파일을 업로드합니다.
- 버킷에서 객체를 다운로드합니다.
- 버킷의 하위 폴더에 객체를 복사합니다.
- 버킷의 객체를 나열합니다.
- 버킷 객체와 버킷을 삭제합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

먼저 필요한 모듈을 모두 가져옵니다.

```
// 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";
```

이전 가져오기는 일부 도우미 유틸리티를 참조합니다. 이러한 유틸리티는 이 섹션의 시작 부분에 링크된 GitHub 저장소에 로컬로 제공됩니다. 참조를 위해 해당 유틸리티의 다음 구현을 참조하십시오.

```
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`;
};
```

S3의 객체는 '버킷'에 저장됩니다. 새 버킷을 만들기 위한 함수를 정의해 보겠습니다.

```
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;
};
```

버킷에는 '객체'가 포함됩니다. 이 함수는 디렉터리의 콘텐츠를 버킷에 객체로 업로드합니다.

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

객체를 업로드한 후 객체가 올바르게 업로드되었는지 확인하십시오. 이를 ListObjects 위해 사용할 수 있습니다. 'Key' 속성을 사용하겠지만 응답에는 다른 유용한 속성도 있습니다.

```
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");
};
```

때로는 한 버킷에서 다른 버킷으로 객체를 복사하고 싶을 수도 있습니다. 그러려면 CopyObject 명령을 사용하세요.

```
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 (err) {
```

```
        console.error(`Copy error.`);
        console.error(err);
        const retryAnswer = await prompter.confirm({ message: "Try again?" });
        if (retryAnswer) {
          await copy();
        }
      }
    };
    await copy();
  }
};
```

버킷에서 여러 객체를 가져오는 SDK 메서드는 없습니다. 대신, 다운로드 및 반복할 객체 목록을 생성하겠습니다.

```
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");
};
```

이제 리소스를 정리할 차례입니다. 삭제하려면 버킷이 비어 있어야 합니다. 이 두 함수는 버킷을 비우고 삭제합니다.

```
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`);
};
```

'main' 함수는 모든 것을 한데 가져옵니다. 이 파일을 직접 실행하면 main 함수가 호출됩니다.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

대용량 파일 업로드 또는 다운로드

다음 코드 예제에서는 Amazon S3에 대용량 파일을 업로드하고 Amazon S3에서 대용량 파일을 다운로드하는 방법을 보여줍니다.

자세한 내용은 멀티파트 업로드를 사용하여 객체 업로드를 참조하십시오.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

대용량 파일을 업로드합니다.

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

대용량 파일을 다운로드합니다.

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

(v3) 용 JavaScript SDK를 사용한 S3 글레이셔 예제

다음 코드 예제는 S3 Glacier와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

작업

볼트 생성

다음 코드 예제에서는 Amazon S3 Glacier 볼트를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

볼트를 생성하세요.

```
// 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();
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateVault참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateVault참조를 참조하십시오.

아카이브를 볼트에 업로드

다음 코드 예제에서는 Amazon S3 Glacier 볼트에 아카이브를 업로드하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

아카이브를 업로드하세요.

```
// 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();
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API UploadArchive참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API UploadArchive참조를 참조하십시오.

SageMaker JavaScript (v3) 용 SDK 사용 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. SageMaker

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

시작하기

안녕하세요. SageMaker

다음 코드 예제에서는 SageMaker의 사용을 시작하는 방법을 보여 줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListNotebookInstances참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

파이프라인 생성

다음 코드 예제는 에서 파이프라인을 생성하거나 업데이트하는 방법을 보여줍니다 SageMaker.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

로컬에서 제공된 JSON 정의를 사용하여 SageMaker 파이프라인을 생성하는 함수입니다.

```
/**
 * 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 }),
      );
    },
  };
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - CreatePipeline
  - UpdatePipeline

파이프라인 삭제

다음 코드 예제는 에서 SageMaker 파이프라인을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SageMaker 파이프라인 삭제 구문. 이 코드는 더 큰 함수의 일부입니다. 자세한 내용은 '파이프라인 만들기' 또는 GitHub 리포지토리를 참조하십시오.

```
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
```

- API 세부 정보는 API 참조를 참조하십시오 DeletePipeline.AWS SDK for JavaScript

파이프라인 실행 설명

다음 코드 예제는 에서 파이프라인 실행을 설명하는 방법을 보여줍니다 SageMaker.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SageMaker 파이프라인 실행이 성공, 실패 또는 중지될 때까지 기다리세요.

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

- API 세부 정보는 AWS SDK for JavaScript API DescribePipelineExecution참조를 참조하십시오.

파이프라인 실행

다음 코드 예제는 에서 파이프라인 실행을 시작하는 방법을 보여줍니다 SageMaker.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SageMaker 파이프라인 실행 시작.

```
/**
 * 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,
  };
}
```

- API 세부 정보는 AWS SDK for JavaScript API 레퍼런스를 참조하십시오 StartPipelineExecution.

시나리오

지리공간 작업 및 파이프라인으로 시작하기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 파이프라인의 리소스를 설정하세요.
- 지리 공간 작업을 실행하는 파이프라인을 설정합니다.
- 파이프라인 실행을 시작합니다.
- 실행 상태를 모니터링합니다.
- 파이프라인의 출력을 볼 수 있습니다.
- 리소스를 정리합니다.

자세한 내용은 Community.AWS에서 AWS SDK를 사용하여 SageMaker 파이프라인 생성 및 실행을 참조하십시오.

(v3) 용 SDK JavaScript

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

다음 파일 발췌문에는 SageMaker 클라이언트를 사용하여 파이프라인을 관리하는 함수가 포함되어 있습니다.

```
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();
}
```

이 함수는 위의 라이브러리 함수를 사용하여 SageMaker 파이프라인을 설정하고 실행하고 생성된 모든 리소스를 삭제하는 파일에서 발췌한 것입니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution

- UpdatePipeline

JavaScript (v3) 용 SDK를 사용한 Secrets Manager 예제

다음 코드 예제는 Secrets Manager와 함께 AWS SDK for JavaScript (v3) 를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업

작업

보안 암호 값 가져오기

다음 코드 예제에서는 Secrets Manager 보안 암호 값을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetSecretValue참조를 참조하십시오.

JavaScript (v3) 용 SDK를 사용하는 Amazon SES 예제

다음 코드 예제는 Amazon SES와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업

작업

수신 필터 생성

다음 코드 예제에서는 IP 주소 또는 IP 주소 범위에서 들어오는 수신 메일을 차단하는 Amazon SES 수신 필터를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  CreateReceiptFilterCommand,
  ReceiptFilterPolicy,
} from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateReceiptFilter참조를 참조하십시오.

수신 규칙 생성

다음 코드 예제에서는 Amazon SES 수신 규칙을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateReceiptRule참조를 참조하십시오.

수신 규칙 세트 생성

다음 코드 예제에서는 수신 메일에 적용할 규칙을 구성하기 위한 Amazon SES 수신 규칙 세트를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateReceiptRuleSet참조를 참조하십시오.

이메일 템플릿 생성

다음 코드 예제에서는 Amazon SES 이메일 템플릿을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateTemplate참조를 참조하십시오.

수신 필터 삭제

다음 코드 예제에서는 Amazon SES 수신 필터를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteReceiptFilter참조를 참조하십시오.

수신 규칙 삭제

다음 코드 예제에서는 Amazon SES 수신 규칙을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteReceiptRule참조를 참조하십시오.

규칙 세트 삭제

다음 코드 예제에서는 Amazon SES 규칙 세트 및 포함된 모든 규칙을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteReceiptRuleSet참조를 참조하십시오.

이메일 템플릿 삭제

다음 코드 예제에서는 Amazon SES 이메일 템플릿을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteTemplate참조를 참조하십시오.

자격 증명 삭제

다음 코드 예제에서는 Amazon SES 자격 증명을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DeleteIdentity참조를 참조하십시오.

기존 이메일 템플릿 가져오기

다음 코드 예제에서는 기존 Amazon SES 이메일 템플릿을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetTemplate참조를 참조하십시오.

이메일 템플릿 나열

다음 코드 예제에서는 Amazon SES 이메일 템플릿을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListTemplates참조를 참조하십시오.

자격 증명 나열

다음 코드 예제에서는 Amazon SES 자격 증명을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListIdentities참조를 참조하십시오.

수신 필터 나열

다음 코드 예제에서는 Amazon SES 수신 필터를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
```

```
};
```

- API 세부 정보는 AWS SDK for JavaScript API ListReceiptFilters참조를 참조하십시오.

대량 템플릿 이메일 전송

다음 코드 예제에서는 Amazon SES를 통해 여러 대상에 템플릿 이메일을 전송하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API SendBulkTemplatedEmail참조를 참조하십시오.

이메일 보내기

다음 코드 예제에서는 Amazon SES를 사용하여 이메일을 전송하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API SendEmail참조를 참조하십시오.

원시 이메일 전송

다음 코드 예제에서는 Amazon SES를 통해 원시 이메일을 전송하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

nodemailer를 사용하여 첨부 파일이 있는 이메일을 보냅니다.

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

- API 세부 정보는 AWS SDK for JavaScript API SendRawEmail참조를 참조하십시오.

템플릿 이메일 전송

다음 코드 예제에서는 Amazon SES를 통해 템플릿 이메일을 전송하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API SendTemplatedEmail참조를 참조하십시오.

이메일 템플릿 업데이트

다음 코드 예제에서는 Amazon SES 이메일 템플릿을 업데이트하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API UpdateTemplate참조를 참조하십시오.

도메인 자격 증명 확인

다음 코드 예제에서는 Amazon SES를 통해 도메인 자격 증명을 확인하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API VerifyDomainIdentity참조를 참조하십시오.

이메일 자격 증명 확인

다음 코드 예제에서는 Amazon SES를 통해 이메일 자격 증명을 확인하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API VerifyEmailIdentity참조를 참조하십시오.

JavaScript (v3) 에 SDK를 사용하는 Amazon SNS 예제

다음 코드 예제는 Amazon SNS와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 상황에 맞게 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

시작하기

Hello Amazon SNS

다음 코드 예시는 Amazon SNS 사용을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SNS 클라이언트를 초기화하고 계정의 주제를 나열하세요.

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

- API 세부 정보는 AWS SDK for JavaScript API ListTopics참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

전화번호가 옵트아웃되었는지 확인

다음 코드 예제는 전화번호가 Amazon SNS 메시지 수신을 옵트아웃했는지 확인하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CheckIfPhoneNumberIsOptedOut참조를 참조하십시오.

엔드포인트 소유자가 메시지를 수신하기를 원하는지 확인

다음 코드 예제에서는 이전 구독 작업을 통해 엔드포인트로 전송된 토큰의 유효성을 검증하여 엔드포인트의 소유자가 Amazon SNS 메시지를 수신하기를 원하는지 확인하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ConfirmSubscription참조를 참조하십시오.

주제 생성

다음 코드 예제는 Amazon SNS 주제를 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateTopic참조를 참조하십시오.

구독 삭제

다음 코드 예제에서는 Amazon SNS 구독을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 Unsubscribe를 참조하세요.

주제 삭제

다음 코드 예제에서는 Amazon SNS 주제 및 해당 주제에 대한 모든 구독을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteTopic참조를 참조하십시오.

주제의 속성 가져오기

다음 코드 예제는 Amazon SNS 주제의 속성을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetTopicAttributes참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 더 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetTopicAttributes참조를 참조하십시오.

SMS 메시지 전송 설정 가져오기

다음 코드 예제는 Amazon SNS SMS 메시지 전송에 대한 설정을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetSMSAttributes를 참조하십시오.

주제의 구독자 나열

다음 코드 예제는 Amazon SNS 주제의 구독자 목록을 검색하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListSubscriptions참조를 참조하십시오.

주제 나열

다음 코드 예제는 Amazon SNS 주제를 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListTopics참조를 참조하십시오.

속성을 사용하여 메시지 게시

다음 코드 예제에서는 Amazon SNS를 통해 속성을 사용하여 메시지를 게시하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

그룹, 복제, 속성 옵션을 사용하여 주제에 메시지를 게시하세요.

```
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();
    }
  }
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Publish를 참조하십시오.

주제 게시

다음 코드 예제에서는 Amazon SNS 주제에 메시지를 게시하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 Publish를 참조하세요.

SMS 메시지 전송 기본값 설정

다음 코드 예제는 Amazon SNS를 사용하여 SMS 메시지를 보내기 위한 기본 설정을 지정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 SetSMSAttributes를 참조하세요.

주제 속성 설정

다음 코드 예제는 Amazon SNS 주제 속성을 설정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API SetTopicAttributes참조를 참조하십시오.

Lambda 함수로 주제 구독

다음 코드 예제에서는 Amazon SNS 주제에서 알림을 수신하도록 Lambda 함수로 구독하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Subscribe를 참조하십시오.

모바일 애플리케이션으로 주제 구독

다음 코드 예제에서는 Amazon SNS 주제에서 알림을 수신하도록 모바일 애플리케이션 엔드포인트로 구독하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Subscribe를 참조하십시오.

SQS 대기열로 주제 구독

다음 코드 예시에서는 Amazon SNS 주제에서 알림을 수신하도록 Amazon SQS 대기열로 구독하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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-xxxxxxxxxxxx'
  // }
  return response;
```

```
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 Subscribe를 참조하세요.

이메일 주소로 주제 구독

다음 코드 예제는 이메일 주소에서 Amazon SNS를 구독하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```
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({});
```

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Subscribe를 참조하십시오.

필터를 사용하여 주제 구독

다음 코드 예시에서는 필터를 사용하여 Amazon SNS 주제를 구독하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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-xxxxxxxxxxxx'
  // }
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Subscribe를 참조하십시오.

시나리오

대기열에 메시지 게시

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 주제(FIFO 또는 비 FIFO)를 생성합니다.
- 필터 적용 옵션을 사용하여 여러 개의 대기열로 주제를 구독합니다.
- 주제에 메시지를 게시합니다.
- 대기열에서 받은 메시지를 폴링합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 워크플로의 시작점입니다.

```
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();
};
```

위 코드는 필요한 종속성을 제공하고 워크플로를 시작합니다. 다음 섹션에는 대부분의 예제가 포함되어 있습니다.

```
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();
    }
```

```
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 게시
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

(v3) 용 JavaScript SDK를 사용하는 Amazon SQS 예제

다음 코드 예제는 Amazon SQS와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 상황에 GitHub 맞게 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

시작하기

Amazon SNS 시작

다음 코드 예제에서는 Amazon SQS를 사용하여 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 클라이언트를 초기화하고 대기열을 나열합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API ListQueues참조를 참조하십시오.

주제

- 작업
- 시나리오

작업

메시지 제한 시간 표시 여부 변경

다음 코드 예제에서는 Amazon SQS 메시지 제한 시간 가시성을 변경하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지를 수신하고 제한 시간 표시 여부를 변경합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ChangeMessageVisibility참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지를 수신하고 제한 시간 표시 여부를 변경합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ChangeMessageVisibility참조를 참조하십시오.

DLQ(Dead Letter Queue) 구성

다음 코드 예제에서는 Amazon SQS에서 DLQ(Dead Letter Queue)로 구성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API SetQueueAttributes참조를 참조하십시오.

대기열 생성

다음 코드 예제에서는 Amazon SQS 대기열을 생성하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 표준 대기열을 생성합니다.

```
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;
};
```

긴 폴링이 있는 Amazon SQS 대기열을 생성합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateQueue참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 표준 대기열을 생성합니다.

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

메시지가 도착하기를 기다리는 Amazon SQS 대기열을 생성합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API CreateQueue참조를 참조하십시오.

대기열에서 메시지 배치 삭제

다음 코드 예제에서는 Amazon SQS 대기열에서 메시지 배치를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for JavaScript API DeleteMessageBatch참조를 참조하십시오.

대기열에서 메시지를 삭제

다음 코드 예제에서는 Amazon SQS 대기열에서 메시지를 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지를 수신하고 삭제합니다.

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

- API 세부 정보는 AWS SDK for JavaScript API DeleteMessage참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지를 수신하고 삭제합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteMessage참조를 참조하십시오.

대기열 삭제

다음 코드 예제에서는 Amazon SQS 대기열을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 삭제합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteQueue참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 삭제합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteQueue참조를 참조하십시오.

대기열의 속성 가져오기

다음 코드 예제에서는 Amazon SQS 대기열의 속성을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API GetQueueAttributes참조를 참조하십시오.

대기열의 URL 가져오기

다음 코드 예제에서는 Amazon SQS 대기열의 URL을 가져오는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열의 URL을 가져옵니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetQueueUrl참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열의 URL을 가져옵니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API GetQueueUrl참조를 참조하십시오.

대기열 나열

다음 코드 예제에서는 Amazon SQS 대기열을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 나열합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListQueues참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 나열합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListQueues참조를 참조하십시오.

대기열에서 메시지 수신

다음 코드 예제에서는 Amazon SQS 대기열에서 메시지를 수신하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열에서 메시지를 수신합니다.

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

긴 폴링 지원을 사용하여 Amazon SQS 대기열에서 메시지를 수신합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API ReceiveMessage참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

긴 폴링 지원을 사용하여 Amazon SQS 대기열에서 메시지를 수신합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ReceiveMessage참조를 참조하십시오.

대기열로 메시지 전송

다음 코드 예제에서는 Amazon SQS 대기열에 메시지를 전송하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열에 메시지를 전송합니다.

```
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;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API SendMessage참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열에 메시지를 전송합니다.

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

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API SendMessage참조를 참조하십시오.

대기열 속성 설정

다음 코드 예제에서는 Amazon SQS 대기열의 속성을 설정하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
};
```

긴 폴링을 사용하도록 Amazon SQS 대기열을 구성합니다.

```
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;
};
```

- API 세부 정보는 AWS SDK for JavaScript API SetQueueAttributes참조를 참조하십시오.

시나리오

대기열에 메시지 게시

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 주제(FIFO 또는 비 FIFO)를 생성합니다.
- 필터 적용 옵션을 사용하여 여러 개의 대기열로 주제를 구독합니다.
- 주제에 메시지를 게시합니다.
- 대기열에서 받은 메시지를 폴링합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 워크플로의 시작점입니다.

```
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();
};
```

위 코드는 필요한 종속성을 제공하고 워크플로를 시작합니다. 다음 섹션에는 대부분의 예제가 포함되어 있습니다.

```
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();
    }
```

```
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 게시
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

JavaScript (v3) 용 SDK를 사용한 Step Functions 예제

다음 코드 예제는 Step Functions와 함께 AWS SDK for JavaScript (v3) 를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업

작업

상태 머신 시작

다음 코드 예제에서는 Step Functions 상태 시스템 실행을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- API 세부 정보는 AWS SDK for JavaScript API StartExecution참조를 참조하십시오.

AWS STS JavaScript (v3) 용 SDK 사용 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. AWS STS

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

주제

- 작업

작업

역할 수임

다음 코드 예제는 에서 역할을 맡는 방법을 보여줍니다 AWS STS.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 });
```

IAM 역할을 수임합니다.

```
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);
```

```
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API AssumeRole참조를 참조하십시오.

JavaScript (v2) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// 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,
    };
    stsGetCallerIdentity(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);
    }
  });
}
```

- API 세부 정보는 AWS SDK for JavaScript API AssumeRole참조를 참조하십시오.

AWS Support JavaScript (v3) 용 SDK 사용 예제

다음 코드 예제는 AWS SDK for JavaScript (v3) 와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다. AWS Support

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

시작하기

안녕하세요. AWS Support

다음 코드 예제에서는 AWS Support의 사용을 시작하는 방법을 보여 줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

`main ()`을 간접적으로 호출하여 예제를 실행합니다.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeServices참조를 참조하십시오.

주제

작업

사례에 통신 추가하기

다음 코드 예제는 첨부 파일이 있는 AWS Support 통신을 지원 사례에 추가하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API AddCommunicationToCase참조를 참조하십시오.

세트에 첨부 파일 추가하기

다음 코드 예제는 AWS Support 어태치먼트 세트에 어태치먼트를 추가하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API AddAttachmentsToSet참조를 참조하십시오.

사례 만들기

다음 코드 예제는 새 AWS Support 사례를 만드는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API CreateCase참조를 참조하십시오.

첨부 파일 설명하기

다음 코드 예제에서는 AWS Support 사례에 대한 첨부 파일을 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeAttachment참조를 참조하십시오.

사례 설명하기

다음 코드 예제는 AWS Support 사례를 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
```

```
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeCases참조를 참조하십시오.

통신 설명하기

다음 코드 예제는 사례에 대한 AWS Support 커뮤니케이션을 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
```

```
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeCommunications참조를 참조하십시오.

심각도 수준 설명하기

다음 코드 예제는 AWS Support 심각도 수준을 설명하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API DescribeSeverityLevels참조를 참조하십시오.

사례 해결

다음 코드 예제는 AWS Support 사례를 해결하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API ResolveCase참조를 참조하십시오.

시나리오

사례 시작하기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 사용 가능한 서비스 및 사례의 심각도 수준을 가져와서 표시합니다.
- 선택한 서비스, 범주 및 심각도 수준을 사용하여 지원 사례를 만듭니다.
- 현재 일자의 미해결 사례 목록을 가져와서 표시합니다.
- 새로운 사례에 첨부 파일 세트와 통신을 추가합니다.

- 해당 사례에 대한 새로운 첨부 파일과 통신을 설명하세요.
- 사건을 해결하세요.
- 현재 일자의 해결된 사례 목록을 가져와서 표시합니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

터미널에서 대화형 시나리오를 실행합니다.

```
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 startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.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 },
        () => getTodaysResolvedCases(caseId)
      );
      console.log("Resolved cases:");
      console.log(resolvedCases.map((c) => c.caseId).join("\n"));
    }
  } catch (err) {
    console.error(err);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment

- DescribeCases
- DescribeCommunications
- DescribeServices
- DescribeSeverityLevels
- ResolveCase

(v3) 용 JavaScript SDK를 사용한 Amazon Transcribe 예제

다음 코드 예제는 Amazon Transcribe와 함께 AWS SDK for JavaScript (v3) 을 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 크로스 서비스 예제에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

각 예제에는 GitHub 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

주제

- 작업

작업

의료 트랜스크립션 작업 삭제

다음 코드 예제에서는 Amazon Transcribe Medical 트랜스크립션 작업을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

의료 트랜스크립션 작업을 삭제합니다.

```
// 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_transciption_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();
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteMedicalTranscriptionJob참조를 참조하십시오.

트랜스크립션 작업 삭제

다음 코드 예제에서는 Amazon Transcribe 트랜스크립션 작업을 삭제하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

트랜스크립션 작업을 삭제합니다.

```
// 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, 'transciption_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();
```

클라이언트를 생성합니다.

```
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 };
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API DeleteTranscriptionJob참조를 참조하십시오.

의료 트랜스크립션 작업 나열

다음 코드 예제에서는 Amazon Transcribe Medical 트랜스크립션 작업을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

의료 트랜스크립션 작업을 나열합니다.

```
// 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();
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListMedicalTranscriptionJobs참조를 참조하십시오.

트랜스크립션 작업 나열

다음 코드 예제에서는 Amazon Transcribe 트랜스크립션 작업을 나열하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

트랜스크립션 작업을 나열합니다.

```
// 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();
```

클라이언트를 생성합니다.

```
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 };
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API ListTranscriptionJobs참조를 참조하십시오.

의료 트랜스크립션 작업 시작

다음 코드 예제에서는 Amazon Transcribe Medical 트랜스크립션 작업을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
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 };
```

의료 트랜스크립션 작업을 시작합니다.

```
// 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();
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API StartMedicalTranscriptionJob참조를 참조하십시오.

트랜스크립션 작업 시작

다음 코드 예제에서는 Amazon Transcribe 트랜스크립션 작업을 시작하는 방법을 보여줍니다.

JavaScript (v3) 용 SDK

> **Note**
>
> 더 많은 내용이 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

트랜스크립션 작업을 시작합니다.

```
// 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();
```

클라이언트를 생성합니다.

```
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 };
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API StartTranscriptionJob참조를 참조하십시오.

JavaScript (v3) 용 SDK를 사용한 크로스 서비스 예제

다음 샘플 애플리케이션은 AWS SDK for JavaScript (v3) 를 사용하여 여러 애플리케이션에서 작동합니다. AWS 서비스

크로스 서비스 예제는 애플리케이션 구축을 시작하는 데 도움이 되는 고급 수준의 경험을 대상으로 합니다.

예

- Amazon Transcribe 앱 구축
- Amazon Transcribe 스트리밍 앱 구축
- DynamoDB 테이블에 데이터를 제출하기 위한 애플리케이션 구축
- Amazon Lex 챗봇을 구축하여 웹 사이트 방문자의 참여 유도
- 사용자가 레이블을 사용하여 사진을 관리할 수 있는 사진 자산 관리 애플리케이션 만들기
- DynamoDB 데이터를 추적하는 웹 애플리케이션 생성
- Aurora 서버리스 작업 항목 트래커 만들기
- Amazon Textract 탐색기 애플리케이션 생성
- 고객 피드백을 분석하고 오디오를 합성하는 애플리케이션 생성
- Amazon Rekognition으로 SDK를 사용하여 이미지에서 PPE를 감지합니다. AWS
- Amazon Rekognition으로 SDK를 사용하여 이미지 내 객체를 감지합니다. AWS
- Amazon Rekognition에서 SDK를 사용하여 동영상 속 사람과 물체를 감지합니다. AWS
- 브라우저에서 Lambda 함수 호출
- API Gateway를 사용하여 Lambda 함수 호출
- Step Functions를 사용하여 Lambda 함수 호출
- 예약된 이벤트를 사용하여 Lambda 함수 호출

Amazon Transcribe 앱 구축

JavaScript (v3) 용 SDK

> Amazon Transcribe를 사용하여 브라우저에서 음성 녹음을 텍스트로 기록하고 표시하는 앱을 만듭니다. 이 앱은 두 개의 Amazon Simple Storage Service(S3) 버킷을 사용합니다. 하나는 애플리케이션 코드를 호스팅하고 다른 하나는 트랜스크립션을 저장합니다. 이 앱은 Amazon Cognito 사용

자 풀을 사용하여 사용자를 인증합니다. 인증된 사용자는 필요한 서비스에 액세스할 수 있는 AWS Identity and Access Management (IAM) 권한을 가집니다. AWS

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오. GitHub

이 예시는 AWS SDK for JavaScript v3 개발자 안내서에서도 확인할 수 있습니다.

이 예시에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon Transcribe

Amazon Transcribe 스트리밍 앱 구축

JavaScript (v3) 용 SDK

Amazon Transcribe를 사용하여 라이브 오디오를 실시간으로 기록, 변환 및 번역하고 Amazon Simple Email Service(Amazon SES)를 사용하여 결과를 이메일로 전송하는 앱을 구축하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 의 전체 예제를 참조하십시오. GitHub

이 예시에서 사용되는 서비스

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

DynamoDB 테이블에 데이터를 제출하기 위한 애플리케이션 구축

JavaScript (v3) 용 SDK

이 예제에서는 사용자가 Amazon DynamoDB 테이블에 데이터를 제출하고 Amazon Simple Notification Service(Amazon SNS)를 사용하여 관리자에게 문자 메시지를 전송하는 앱을 구축하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 의 전체 예제를 참조하십시오. GitHub

이 예시는 AWS SDK for JavaScript v3 개발자 안내서에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SNS

Amazon Lex 챗봇을 구축하여 웹 사이트 방문자의 참여 유도

JavaScript (v3) 용 SDK

Amazon Lex API를 사용하여 웹 애플리케이션 내에서 챗봇을 생성하여 웹 사이트 방문자를 참여시키는 방법을 보여줍니다.

전체 소스 코드와 설치 및 실행 방법에 대한 지침은 AWS SDK for JavaScript 개발자 안내서의 Amazon Lex 챗봇 구축 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

사용자가 레이블을 사용하여 사진을 관리할 수 있는 사진 자산 관리 애플리케이션 만들기

JavaScript (v3) 용 SDK

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 의 전체 예제를 참조하십시오. GitHub

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하십시오.

이 예시에서 사용되는 서비스

- API Gateway
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

DynamoDB 데이터를 추적하는 웹 애플리케이션 생성

JavaScript (v3) 용 SDK

Amazon DynamoDB API를 사용하여 DynamoDB 작업 데이터를 추적하는 동적 웹 애플리케이션을 생성하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 의 전체 예제를 참조하십시오. GitHub

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SES

Aurora 서버리스 작업 항목 트래커 만들기

JavaScript (v3) 용 SDK

AWS SDK for JavaScript (v3) 를 사용하여 Amazon Aurora 데이터베이스의 작업 항목을 추적하고 Amazon Simple Email Service (Amazon SES) 를 사용하여 보고서를 이메일로 보내는 웹 애플리케이션을 생성하는 방법을 보여 줍니다. 이 예제에서는 Express Node.js 백엔드와의 상호 작용을 위해 React.js로 빌드된 프런트엔드를 사용합니다.

- React.js 웹 애플리케이션을 와 통합하십시오. AWS 서비스
- Aurora 테이블의 항목을 나열, 추가 및 업데이트합니다.
- Amazon SES를 사용하여 필터링된 작업 항목에 대한 이메일 보고서를 보냅니다.
- 포함된 AWS CloudFormation 스크립트를 사용하여 예제 리소스를 배포하고 관리합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 의 전체 예제를 참조하십시오 GitHub.

이 예시에서 사용되는 서비스

- Aurora

- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

Amazon Textract 탐색기 애플리케이션 생성

JavaScript (v3) 용 SDK

를 사용하여 Amazon AWS SDK for JavaScript Textract를 사용하여 문서 이미지에서 데이터를 추출하고 대화형 웹 페이지에 표시하는 React 애플리케이션을 구축하는 방법을 보여 줍니다. 이 예제는 웹 브라우저에서 실행되며 자격 증명을 위해 인증된 Amazon Cognito 자격 증명이 필요합니다. 이 애플리케이션은 스토리지로 Amazon Simple Storage Service(S3)를 사용하고 알림을 위해 Amazon Simple Notification Service(Amazon SNS) 주제를 구독하는 Amazon Simple Queue Service(Amazon SQS) 대기열을 폴링합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오. GitHub

이 예시에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

고객 피드백을 분석하고 오디오를 합성하는 애플리케이션 생성

JavaScript (v3) 용 SDK

이 예제 애플리케이션은 고객 피드백 카드를 분석하고 저장합니다. 특히 뉴욕시에 있는 가상 호텔의 필요를 충족합니다. 호텔은 다양한 언어의 고객들로부터 물리적인 의견 카드의 형태로 피드백을 받습니다. 피드백은 웹 클라이언트를 통해 앱에 업로드됩니다. 의견 카드의 이미지가 업로드된 후 다음 단계가 수행됩니다.

- Amazon Textract를 사용하여 이미지에서 텍스트가 추출됩니다.
- Amazon Comprehend가 추출된 텍스트와 해당 언어의 감정을 파악합니다.

- 추출된 텍스트는 Amazon Translate를 사용하여 영어로 번역됩니다.
- Amazon Polly가 추출된 텍스트에서 오디오 파일을 합성합니다.

전체 앱은 AWS CDK를 사용하여 배포할 수 있습니다. 소스 코드 및 배포 지침은 에서 프로젝트를 참조하십시오. GitHub 다음 발췌문은 Lambda 함수 내에서 AWS SDK for JavaScript 가 사용되는 방법을 보여줍니다.

```
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 };
};
```

이 예시에서 사용되는 서비스

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Amazon Rekognition으로 SDK를 사용하여 이미지에서 PPE를 감지합니다. AWS

(v3) 용 SDK JavaScript

Amazon AWS SDK for JavaScript Rekognition과 함께 사용하여 Amazon Simple Storage Service (Amazon S3) 버킷에 있는 이미지에서 개인 보호 장비 (PPE) 를 탐지하는 애플리케이션을 만드는 방법을 보여 줍니다. 이 앱은 결과를 Amazon DynamoDB 테이블에 저장하고 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

다음 작업을 수행하는 방법에 대해 알아보세요.

- Amazon Cognito를 사용하여 인증되지 않은 사용자를 생성합니다.

- Amazon Rekognition을 사용하여 PPE용 이미지를 분석합니다.
- Amazon SES 이메일 주소를 확인합니다.
- DynamoDB 테이블을 결과로 업데이트합니다.
- Amazon SES를 사용하여 이메일 알림을 전송합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오. GitHub

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Amazon Rekognition으로 SDK를 사용하여 이미지 내 객체를 감지합니다. AWS

(v3) 용 SDK JavaScript

Amazon Rekognition을 AWS SDK for JavaScript 와 함께 사용하여 Amazon Simple Storage Service (Amazon S3) 버킷에 있는 이미지의 범주별로 객체를 식별하는 앱을 만드는 방법을 보여 줍니다. 이 앱은 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

다음 작업을 수행하는 방법에 대해 알아보십시오.

- Amazon Cognito를 사용하여 인증되지 않은 사용자를 생성합니다.
- Amazon Rekognition을 사용하여 객체용 이미지를 분석합니다.
- Amazon SES 이메일 주소를 확인합니다.
- Amazon SES를 사용하여 이메일 알림을 전송합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오. GitHub

이 예시에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3

- Amazon SES

Amazon Rekognition에서 SDK를 사용하여 동영상 속 사람과 물체를 감지합니다. AWS

(v3) 용 SDK JavaScript

Amazon AWS SDK for JavaScript Rekognition과 함께 사용하여 Amazon Simple Storage Service (Amazon S3) 버킷에 있는 비디오에서 얼굴과 사물을 감지하는 앱을 만드는 방법을 보여 줍니다. 이 앱은 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

다음 작업을 수행하는 방법에 대해 알아보십시오.

- Amazon Cognito를 사용하여 인증되지 않은 사용자를 생성합니다.
- Amazon Rekognition을 사용하여 PPE용 이미지를 분석합니다.
- Amazon SES 이메일 주소를 확인합니다.
- Amazon SES를 사용하여 이메일 알림을 전송합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오. GitHub

이 예시에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

브라우저에서 Lambda 함수 호출

JavaScript (v2) 용 SDK

AWS Lambda 함수를 사용하여 Amazon DynamoDB 테이블을 사용자 선택으로 업데이트하는 브라우저 기반 애플리케이션을 생성할 수 있습니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오. GitHub

이 예제에서 사용되는 서비스

- DynamoDB

- Lambda

JavaScript (v3) 용 SDK

AWS Lambda 함수를 사용하여 Amazon DynamoDB 테이블을 사용자 선택으로 업데이트하는 브라우저 기반 애플리케이션을 생성할 수 있습니다. 이 앱은 v3을 사용합니다. AWS SDK for JavaScript

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 의 전체 예제를 참조하십시오. GitHub

이 예제에서 사용되는 서비스

- DynamoDB
- Lambda

API Gateway를 사용하여 Lambda 함수 호출

JavaScript (v3) 용 SDK

JavaScript Lambda AWS Lambda 런타임 API를 사용하여 함수를 생성하는 방법을 보여 줍니다. 이 예제는 다양한 AWS 서비스를 호출하여 특정 사용 사례를 수행합니다. 이 예제에서는 Amazon API Gateway에서 호출한 Lambda 함수를 생성하여 작업 기념일에 대한 Amazon DynamoDB 테이블을 스캔하고 Amazon Simple Notification Service(Amazon SNS)를 사용하여 직원에게 1주년 기념일을 축하하는 문자 메시지를 전송하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오. GitHub

이 예시는 AWS SDK for JavaScript v3 개발자 안내서에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Step Functions를 사용하여 Lambda 함수 호출

JavaScript (v3) 용 SDK

및 를 사용하여 AWS AWS Step Functions 서버리스 워크플로를 만드는 방법을 보여 줍니다. AWS SDK for JavaScript각 워크플로 단계는 AWS Lambda 함수를 사용하여 구현됩니다.

Lambda는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있게 하는 컴퓨팅 서비스입니다. Step Functions는 Lambda 함수와 기타 AWS 서비스를 결합할 수 있는 서버리스 오케스트레이션 서비스로, 비즈니스 크리티컬 애플리케이션을 구축합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오 GitHub.

이 예시는 AWS SDK for JavaScript v3 개발자 안내서에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

예약된 이벤트를 사용하여 Lambda 함수 호출

JavaScript (v3) 용 SDK

AWS Lambda 함수를 호출하는 Amazon EventBridge 예약 이벤트를 생성하는 방법을 보여 줍니다. cron 표현식을 사용하여 Lambda 함수가 호출되는 시기를 EventBridge 스케줄링하도록 구성합니다. 이 예시에서는 Lambda 런타임 API를 사용하여 Lambda 함수를 생성합니다. JavaScript 이 예제는 다양한 AWS 서비스를 호출하여 특정 사용 사례를 수행합니다. 이 예제에서는 1주년 기념일에 직원에게 축하하는 모바일 문자 메시지를 전송하는 앱을 생성하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 전체 예제를 참조하십시오. GitHub

이 예시는 AWS SDK for JavaScript v3 개발자 안내서에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

이 AWS 제품 또는 서비스의 보안

Amazon Web Services(AWS)에서 가장 우선순위가 높은 것이 클라우드 보안입니다. AWS 고객은 보안에 매우 민감한 조직의 요건에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다. 보안은 사용자와 사용자 간의 AWS 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

클라우드 보안 — AWS 클라우드에서 제공되는 모든 서비스를 실행하는 인프라를 보호하고 안전하게 사용할 수 있는 서비스를 제공하는 역할을 합니다. AWS 당사의 보안 책임은 최우선 과제이며 AWS, AWS 규정 준수 프로그램의 일환으로 타사 감사자가 보안 효과를 정기적으로 테스트하고 검증합니다.

클라우드에서의 보안 — 사용자의 책임은 사용 중인 AWS 서비스와 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요인에 따라 결정됩니다.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

주제

이 AWS 제품 또는 서비스의 데이터 보호

AWS 공동 책임 모델 이 AWS 제품 또는 서비스의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 은 (는) 모두를 실행하는 글로벌 인프라를 보호할 책임이 AWS 클라우드있습니다. 이 인프라에서 호스팅되는 콘텐츠에 대한 제어를 유지하는 것은 사용자의 책임입니다. 사용하는 AWS 서비스 의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 AWS 계정 자격 증명을 보호하고 AWS IAM Identity Center OR AWS Identity and Access Management (IAM) 을 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이 방식을 사용하면 각 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 리소스와 통신할 수 있습니다. AWS TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다. AWS CloudTrail
- 포함된 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용하십시오 AWS 서비스.
- Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하여 Amazon S3에 저장된 민감한 데이터를 검색하고 보호합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해 액세스할 때 FIPS 140-2로 검증된 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용하십시오. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-2를 참조하십시오.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 콘솔, API 또는 SDK를 AWS 서비스 사용하여 이 AWS 제품이나 서비스 또는 기타 제품을 사용하는 경우가 포함됩니다. AWS CLI AWS 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 보안 인증 정보를 URL에 포함시켜서는 안 됩니다.

ID 및 액세스 관리

AWS Identity and Access Management (IAM) 은 관리자가 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 AWS 도와줍니다. IAM 관리자는 리소스를 사용할 수 있는 인증 (로그인) 및 권한 부여 (권한 보유) 를 받을 수 있는 사용자를 제어합니다. AWS IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

주제

- 고객
- 자격 증명을 통한 인증
- 정책을 사용한 액세스 관리
- AWS 서비스 IAM을 사용하는 방법

- AWS ID 및 액세스 문제 해결

고객

사용하는 방식 AWS Identity and Access Management (IAM) 은 수행하는 작업에 따라 다릅니다. AWS

서비스 사용자 - 작업을 수행하는 AWS 서비스 데 사용하는 경우 관리자가 필요한 자격 증명과 권한을 제공합니다. 더 많은 AWS 기능을 사용하여 작업을 수행함에 따라 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. 에서 AWS기능에 액세스할 수 없는 경우 사용 중인 기능의 사용 설명서를 참조하십시오AWS ID 및 액세스 문제 해결. AWS 서비스

서비스 관리자 — 회사에서 AWS 리소스를 담당하는 경우 전체 액세스 권한이 있을 수 AWS있습니다. 서비스 사용자가 액세스해야 하는 AWS 기능과 리소스를 결정하는 것은 여러분의 몫입니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사에서 IAM을 어떻게 사용할 수 있는지 자세히 알아보려면 사용 중인 사용 설명서를 참조하십시오. AWS AWS 서비스

IAM 관리자 - IAM 관리자라면 AWS에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 AWS ID 기반 정책의 예를 보려면 사용 중인 사용 설명서를 참조하십시오. AWS 서비스

자격 증명을 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페더레이션 ID의 예입니다. 연동 자격 증명으로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 내 로그인 방법을 참조하십시오. AWS AWS 계정

AWS 프로그래밍 방식으로 액세스하는 경우 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트 (SDK) 와 명령줄 인터페이스 (CLI) 를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청 서명을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다중 인증 및 IAM 사용자 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하세요.

AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 태스크의 전체 목록은 IAM 사용자 설명서의 루트 사용자 보안 인증이 필요한 태스크를 참조하세요.

연동 보안 인증

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 비롯한 수동 AWS 서비스 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 액세스하도록 하는 것입니다.

페더레이션 ID는 기업 사용자 디렉토리, 웹 ID 공급자, Identity Center 디렉터리의 사용자 또는 ID 소스를 통해 제공된 자격 증명을 사용하여 액세스하는 AWS 서비스 모든 사용자를 말합니다. AWS Directory Service페더레이션 ID에 AWS 계정액세스하면 이들이 역할을 맡고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 자체 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화하여 모든 사용자 및 애플리케이션에서 사용할 수 있습니다. AWS 계정 IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇입니까?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한을 AWS 계정 가진 사용자 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용자 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 보안 인증입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용

자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증 정보만 제공합니다. 자세한 정보는 IAM 사용자 설명서의 IAM 사용자를 만들어야 하는 경우(역할이 아님)를 참조하세요.

IAM 역할

IAM 역할은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console 전환하여 에서 일시적으로 IAM 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용자 설명서의 IAM 역할 사용을 참조하세요.

임시 보안 인증 정보가 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 연동 자격 증명에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 연동 자격 증명이 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용자 설명서의 Creating a role for a third-party Identity Provider(서드 파티 자격 증명 공급자의 역할 만들기) 부분을 참조하세요. IAM 자격 증명 센터를 사용하는 경우 권한 집합을 구성합니다. 인증 후 아이덴티티가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관 짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트 섹션을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 교차 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용자 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.
- 서비스 간 액세스 — 일부는 다른 AWS 서비스서비스의 기능을 AWS 서비스 사용합니다. 예를 들어 서비스에서 직접 호출을 수행하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 태스크를 수행할 수 있습니다.

- 순방향 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 보안 AWS 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 AWS 서비스서비스에 AWS 서비스 요청하기 위한 요청과 결합하여 사용합니다. FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용자 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요.
- 서비스 연결 역할 — 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

- Amazon EC2에서 실행되는 애플리케이션 — IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증 정보를 얻을 수 있습니다. 자세한 정보는 IAM 사용자 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용자 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하세요.

정책을 사용한 액세스 관리

정책을 생성하고 이를 AWS ID 또는 리소스에 AWS 연결하여 액세스를 제어할 수 있습니다. 정책은 ID 또는 리소스와 연결될 때 AWS 해당 권한을 정의하는 객체입니다. AWS 주도자 (사용자, 루트 사용자 또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책의 권한이 요청 허용 또는 거부 여부를 결정합니다. 대부분의 정책은 JSON 문서로 AWS 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용자 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI, 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

보안 인증 기반 정책

보안 인증 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 보안 인증에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용자 설명서의 IAM 정책 생성을 참조하세요.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 내 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용자 설명서의 관리형 정책과 인라인 정책의 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. IAM의 AWS 관리형 정책은 리소스 기반 정책에 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 설명서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ACL을 지원하는 서비스의 예로는 아마존 S3와 아마존 VPC가 있습니다. AWS WAF ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 ACL(액세스 제어 목록) 개요 섹션을 참조하세요.

기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 타입에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 자격 증명 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 특성입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 보안 인증 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 보안 주체로 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용자 설명서의 IAM 엔터티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책 (SCP) - SCP는 조직 또는 조직 단위 (OU) 에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations AWS Organizations 사업체가 소유한 여러 AWS 계정 개를 그룹화하고 중앙에서 관리하는 서비스입니다. 조직에서 모든 특성을 활성화할 경우 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 구성원 계정의 엔티티 (각 엔티티 포함) 에 대한 권한을 제한합니다. AWS 계정 루트 사용자조직 및 SCP에 대한 자세한 정보는 AWS Organizations 사용 설명서의 SCP 작동 방식을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 자격 증명 기반 정책의 교집합과 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용자 설명서의 세션 정책을 참조하세요.

여러 정책 타입

여러 정책 타입이 요청에 적용되는 경우 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련되어 있을 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하십시오.

AWS 서비스 IAM을 사용하는 방법

대부분의 IAM 기능을 어떻게 AWS 서비스 사용하는지 자세히 알아보려면 IAM 사용 설명서의 IAM과 연동되는AWS 서비스를 참조하십시오.

특정 기능을 AWS 서비스 IAM과 함께 사용하는 방법을 알아보려면 관련 서비스 사용 설명서의 보안 섹션을 참조하십시오.

AWS ID 및 액세스 문제 해결

다음 정보를 사용하면 및 IAM을 사용할 때 발생할 수 있는 일반적인 문제를 AWS 진단하고 해결하는 데 도움이 됩니다.

주제

저는 다음과 같은 작업을 수행할 권한이 없습니다. AWS

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음 예제 오류는 mateojackson IAM 사용자가 콘솔을 사용하여 가상 *my-example-widget* 리소스에 대한 세부 정보를 보려고 하지만 가상 awes:*GetWidget* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

이 경우 awes:*GetWidget* 작업을 사용하여 *my-example-widget* 리소스에 액세스할 수 있도록 mateojackson 사용자 정책을 업데이트해야 합니다.

도움이 필요하면 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

저는 IAM을 수행할 권한이 없습니다. PassRole

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

새 서비스 역할 또는 서비스 연결 역할을 만드는 대신 기존 역할을 해당 서비스에 전달할 AWS 서비스 수 있는 기능도 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 AWS에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우 Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요하면 관리자에게 문의하세요. AWS 관리자는 로그인 자격 증명을 제공한 사람입니다.

외부 사용자가 내 AWS 리소스에 액세스할 수 있도록 AWS 계정 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- 이러한 기능의 AWS 지원 여부를 알아보려면 을 참조하십시오AWS 서비스 IAM을 사용하는 방법.
- 소유한 리소스에 대한 액세스 권한을 AWS 계정 부여하는 방법을 알아보려면 IAM 사용 설명서에서 자신이 소유한 다른 AWS 계정 IAM 사용자에게 액세스 권한 제공을 참조하십시오.
- 제3자에게 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정AWS 계정 소유에 대한 액세스 제공을 참조하십시오.
- 자격 증명 연동을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용자 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(자격 증명 연동)을 참조하세요.
- 교차 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용자 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.

이 AWS 제품 또는 서비스에 대한 규정 준수 검증

특정 규정 준수 프로그램의 범위 내에 AWS 서비스 있는지 알아보려면AWS 서비스 규정 준수 프로그램의AWS 서비스 범위별, 규정 참조하여 관심 있는 규정 준수 프로그램을 선택하십시오. 일반 정보는 AWS 규정 준수 프로그램AWS 보증 프로그램 규정AWS 참조하십시오.

를 사용하여 AWS Artifact타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html 참조하십시오 AWS Artifact.

사용 시 규정 준수 AWS 서비스 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- 보안 및 규정 준수 퀵 스타트 가이드 - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 AWS 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계 — 이 백서에서는 기업이 HIPAA 적격 애플리케이션을 만드는 AWS 데 사용할 수 있는 방법을 설명합니다.

> **Note**
>
> 모든 AWS 서비스 사람이 HIPAA 자격을 갖춘 것은 아닙니다. 자세한 내용은 HIPAA 적격 서비스 참조를 참조하십시오.

- AWS 규정 준수 리소스AWS — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 — 규정 준수의 관점에서 공동 책임 모델을 이해하십시오. 이 가이드에서는 보안을 유지하기 위한 모범 사례를 AWS 서비스 요약하고 여러 프레임워크 (미국 표준 기술 연구소 (NIST), 결제 카드 산업 보안 표준 위원회 (PCI), 국제 표준화기구 (ISO) 등) 에서 보안 제어에 대한 지침을 매핑합니다.
- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 — 이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub— 이를 AWS 서비스 통해 내부 AWS보안 상태를 포괄적으로 파악할 수 있습니다. Security Hub는 보안 통제를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하십시오.
- AWS Audit Manager— 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 단순화할 수 있습니다.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

이 AWS 제품 또는 서비스에 대한 복원력

AWS 글로벌 인프라는 가용 영역을 중심으로 AWS 리전 구축됩니다.

AWS 리전 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다. 이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹으로 연결됩니다.

가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 지역 및 가용 영역에 대한 자세한 내용은 글로벌 인프라를 참조하십시오AWS .

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

이 AWS 제품 또는 서비스의 인프라 보안

이 AWS 제품 또는 서비스는 관리 서비스를 사용하므로 AWS 글로벌 네트워크 보안의 보호를 받습니다. AWS 보안 서비스 및 인프라 AWS 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하십시오. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 Security Pillar AWS Well-Architected Framework의 인프라 보호를 참조하십시오.

AWS 게시된 API 호출을 사용하여 네트워크를 통해 이 AWS 제품 또는 서비스에 액세스할 수 있습니다. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 비밀 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 인증 정보를 생성하여 요청에 서명할 수 있습니다.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

최소 TLS 버전 적용

AWS 서비스와 통신할 때 보안을 강화하려면 TLS 1.2 AWS SDK for JavaScript 이상을 사용하도록 구성하십시오.

> **Important**
>
> AWS SDK for JavaScript v3는 해당 서비스 엔드포인트에서 지원하는 최고 수준의 TLS 버전을 자동으로 협상합니다. AWS 선택적으로 애플리케이션에 필요한 최소 TLS 버전 (예: TLS 1.2 또는 1.3) 을 적용할 수 있지만, 일부 AWS 서비스 엔드포인트는 TLS 1.3을 지원하지 않으므로 TLS 1.3을 적용하면 일부 호출이 실패할 수 있다는 점에 유의하십시오.

전송 계층 보안(TLS)은 웹 브라우저 및 기타 애플리케이션에서 네트워크를 통해 교환되는 데이터의 프라이버시 및 무결성을 보장하기 위해 사용하는 프로토콜입니다.

Node.js에서 TLS 확인 및 적용

Node.js AWS SDK for JavaScript 와 함께 사용하는 경우 기본 Node.js 보안 계층을 사용하여 TLS 버전을 설정합니다.

Node.js 12.0.0 이상에서는 TLS 1.3을 지원하는 OpenSSL 1.1.1b의 최소 버전을 사용합니다. AWS SDK for JavaScript v3에서는 사용 가능한 경우 기본적으로 TLS 1.3을 사용하지만 필요한 경우 더 낮은 버전으로 기본 설정됩니다.

OpenSSL 및 TLS의 버전 확인

컴퓨터에 Node.js에서 사용하는 OpenSSL의 버전을 얻으려면 다음 명령을 실행합니다.

```
node -p process.versions
```

목록에 있는 OpenSSL 버전은 다음 예제와 같이 Node.js에서 사용하는 버전입니다.

```
openssl: '1.1.1b'
```

컴퓨터에서 Node.js에서 사용하는 TLS 버전을 얻으려면 노드 셸을 시작하고 순서대로 다음 명령을 실행합니다.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

마지막 명령은 다음 예제와 같이 TLS 버전을 출력합니다.

```
'TLSv1.3'
```

Node.js는 기본적으로 이 버전의 TLS를 사용하고 호출이 성공하지 못하면 다른 버전의 TLS를 협상하려고 시도합니다.

TLS의 최소 버전 적용

Node.js는 호출이 실패하면 TLS 버전을 협상합니다. 이 협상 중에 명령줄에서 스크립트를 실행할 때 또는 코드의 요청에 따라 허용되는 최소 TLS 버전을 적용할 수 있습니다. JavaScript

명령줄에서 최소 TLS 버전을 지정하려면 Node.js 버전 11.0.0 이상을 사용해야 합니다. 특정 Node.js 버전을 설치하려면 먼저, Node version manager installing and updating에 있는 단계를 사용하여 Node Version Manager(nvm)를 설치합니다. 그런 다음, 다음 명령을 실행하여 특정 버전의 Node.js를 설치하고 사용합니다.

```
nvm install 11
nvm use 11
```

Enforce TLS 1.2

TLS 1.2가 허용 가능한 최소 버전인 경우 이를 적용하려면 다음 예제와 같이 스크립트를 실행할 때 `--tls-min-v1.2` 인수를 지정합니다.

```
node --tls-min-v1.2 yourScript.js
```

JavaScript 코드에서 특정 요청에 허용되는 최소 TLS 버전을 지정하려면 다음 예와 같이 `httpOptions` 파라미터를 사용하여 프로토콜을 지정하십시오.

```
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

TLS 1.3이 허용 가능한 최소 버전인 경우 이를 적용하려면 다음 예제와 같이 스크립트를 실행할 때 `--tls-min-v1.3` 인수를 지정합니다.

```
node --tls-min-v1.3 yourScript.js
```

JavaScript 코드에서 특정 요청에 허용되는 최소 TLS 버전을 지정하려면 다음 예제와 같이 `httpOptions` 파라미터를 사용하여 프로토콜을 지정하십시오.

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

브라우저 스크립트에서 TLS 확인 및 적용

브라우저 스크립트에서 SDK for를 JavaScript 사용하는 경우 브라우저 설정에 따라 사용되는 TLS 버전이 제어됩니다. 브라우저에서 사용하는 TLS 버전은 스크립트로 검색하거나 설정할 수 없으며 사용자가 구성해야 합니다. 브라우저 스크립트에 사용된 TLS 버전을 확인하고 적용하려면 해당 브라우저의 지침을 참조하세요.

Microsoft Internet Explorer

1. Internet Explorer를 엽니다.
2. 메뉴 모음에서 도구 – 인터넷 옵션 – 고급 탭을 선택합니다.

3. 보안 범주까지 아래로 스크롤하여 TLS 1.2 사용 옵션 상자를 수동으로 선택합니다.
4. 확인을 클릭합니다.
5. 브라우저를 닫고 Internet Explorer를 다시 시작합니다.

Microsoft Edge

1. Windows 메뉴 검색 상자에 ### ##을 입력합니다.
2. 가장 일치하는 항목에서 인터넷 옵션을 클릭합니다.
3. 인터넷 속성 창의 고급 탭에서 보안 섹션까지 아래로 스크롤합니다.
4. 사용자 TLS 1.2 확인란을 선택합니다.
5. 확인을 클릭합니다.

Google Chrome

1. Google Chrome을 엽니다.
2. Alt F를 클릭하고 설정을 선택합니다.
3. 아래로 스크롤하여 고급 설정 표시를 선택합니다.
4. 시스템 섹션까지 아래로 스크롤하여 프록시 설정 열기를 클릭합니다.
5. 고급 탭을 선택합니다.
6. 보안 범주까지 아래로 스크롤하여 TLS 1.2 사용 옵션 상자를 수동으로 선택합니다.
7. 확인을 클릭합니다.
8. 브라우저를 닫고 Google Chrome을 다시 시작합니다.

Mozilla Firefox

1. Firefox를 엽니다.
2. 주소 표시줄에 about:config를 입력하고 Enter 키를 누릅니다.
3. 검색 필드에 tls를 입력합니다. security.tls.version.min의 항목을 찾아 두 번 클릭합니다.
4. 정수 값을 3으로 설정하여 TLS 1.2의 프로토콜을 기본값으로 강제 설정합니다.
5. 확인을 클릭합니다.
6. 브라우저를 닫고 Mozilla Firefox를 다시 시작합니다.

Apple Safari

SSL 프로토콜을 활성화할 수 있는 옵션은 없습니다. Safari 버전 7 이상을 사용하는 경우 TLS 1.2가 자동으로 활성화됩니다.

버전 3으로 마이그레이션

이 섹션에서는 버전 2에서 버전 3으로 마이그레이션하는 방법을 설명합니다. AWS SDK for JavaScript

코드를 V3용 JavaScript SDK로 마이그레이션하세요.

AWS SDK for JavaScript 버전 3 (v3) 에는 자격 증명, Amazon S3 멀티파트 업로드, DynamoDB 문서 클라이언트, 웨이터 등을 포함하는 클라이언트 구성 및 유틸리티를 위한 현대화된 인터페이스가 함께 제공됩니다. 리포지토리의 마이그레이션 가이드에서 각 변경 사항에 대한 v2 및 v3의 해당 변경 내용을 확인할 수 있습니다. AWS SDK for JavaScript GitHub

AWS SDK for JavaScript v3를 최대한 활용하려면 아래에 설명된 codemod 스크립트를 사용하는 것이 좋습니다.

codemod를 사용하여 기존 v2 코드를 마이그레이션하십시오.

의 codemod 스크립트 컬렉션은 기존 AWS SDK for JavaScript (v2) 애플리케이션을 v3 API를 사용하도록 마이그레이션하는 aws-sdk-js-codemod데 도움이 됩니다. 다음과 같이 변환을 실행할 수 있습니다.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

예를 들어 v2에서 Amazon DynamoDB 클라이언트를 생성하고 `listTables` 작업을 직접적으로 호출하는 다음 코드가 있다고 가정해 보겠습니다.

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

다음과 같이 `example.ts`에서 `v2-to-v3` 변환을 실행할 수 있습니다.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

이 변환은 DynamoDB import를 v3로 변환하고 v3 클라이언트를 생성하며 다음과 같이 `listTables` 작업을 직접적으로 호출합니다.

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

일반적인 사용 사례에 대한 변환을 구현했습니다. 코드가 올바르게 변환되지 않는 경우 입력 코드 예와 관찰/예상된 출력 코드가 포함된 bug report 또는 feature request를 작성하세요. 특정 사용 사례가 existing issue에서 이미 보고된 경우 공감을 표시하여 지지를 보여주세요.

AWS SDK for JavaScript 버전 3의 문서 기록

문서 기록

다음 표는 2020년 10월 20일 이후 AWS SDK for JavaScript V3 릴리스에서 변경된 중요 사항에 대해 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS feed를 구독하세요.

변경 사항	설명	날짜
관련 공지 사항	인터넷 익스플로러 11에 대한 end-of-support 알림으로 상단 배너를 업데이트했습니다.	2022년 9월 23일
마이너 업데이트	끊어진 링크 해결 및 명확성을 위한 마이너 업데이트. AWS SDK 및 도구 참조 가이드에 대한 인식 링크가 추가되었습니다.	2022년 8월 22일
최소 TLS 버전 적용	TLS 1.3에 관한 정보가 추가되었습니다.	2022년 3월 31일
튜토리얼 업데이트 AWS Lambda	Amazon DynamoDB 테이블에 데이터를 제출하기 위해 브라우저 기반 애플리케이션을 빌드하는 방법을 보여주는 자습서가 추가되었습니다.	2020년 10월 20일
Node.js 주제의 자격 증명 설정 (업데이트)	AWS SDK for JavaScript V3용 Node.js 자격 증명을 설정하는 방법에 대한 항목을 업데이트했습니다.	2020년 10월 20일
V3로 마이그레이션	AWS SDK for JavaScript V3로 마이그레이션하는 방법을 설명하는 항목이 추가되었습니다.	2020년 10월 20일

시작하기	브라우저에서 시작하기 및 AWS SDK for JavaScript V3용 Node.js 시작하기 관련 항목이 업데이트되었습니다.	2020년 10월 20일
브라우저 빌더	AWS 브라우저 빌더에 대한 정보는 AWS SDK for JavaScript V3에 필요하지 않으므로 삭제되었습니다.	2020년 10월 20일
Amazon Transcribe 서비스 예 업데이트	V3용 AWS SDK for JavaScript Amazon Transcribe 서비스 예제가 업데이트되었습니다.	2020년 10월 20일
Amazon Simple Notification Service 서비스 예 업데이트	AWS SDK for JavaScript V3용 Amazon 단순 알림 서비스 서비스 예제가 업데이트되었습니다.	2020년 10월 20일
Amazon Simple Email Service 서비스 예 업데이트	AWS SDK for JavaScript V3용 Amazon 단순 이메일 서비스 서비스 예제가 업데이트되었습니다.	2020년 10월 20일
Amazon Redshift 서비스 예 업데이트	V3용 AWS SDK for JavaScript Amazon Redshift 서비스 예제가 업데이트되었습니다.	2020년 10월 20일
Amazon Lex 서비스 예 업데이트	AWS SDK for JavaScript V3용 Amazon Lex 서비스 예제가 업데이트되었습니다.	2020년 10월 20일
Amazon DynamoDB 서비스 예 업데이트	V3용 Amazon DynamoDB 서비스 예제가 업데이트되었습니다. AWS SDK for JavaScript	2020년 10월 20일

AWS Elemental MediaConvert 서비스 예제가 업데이트되었습니다.	AWS SDK for JavaScript V3의 AWS Elemental MediaConvert 서비스 예제가 업데이트되었습니다.	2020년 10월 20일
AWS Lambda 서비스 예제 업데이트	AWS SDK for JavaScript V3의 AWS Lambda 서비스 예제가 업데이트되었습니다.	2020년 10월 20일
AWS SDK for JavaScript V3 개발자 가이드 미리 보기	AWS SDK for JavaScript V3 개발자 안내서의 시험판 버전이 출시되었습니다.	2020년 10월 19일