



SDK 버전 3용 개발자 안내서

AWS SDK for JavaScript



AWS SDK for JavaScript: SDK 버전 3용 개발자 안내서

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

.....	xii
AWS SDK for JavaScript란 무엇인가요?	1
SDK 시작하기	1
SDK 메이저 버전에 대한 유지 관리 및 지원	2
Node.js에서 SDK 사용	2
AWS Amplify에서 SDK 사용	2
웹 브라우저에서 SDK 사용	2
V3에서 브라우저 사용	3
일반 사용 사례	3
예시 관련 정보	4
리소스	4
시작하기	5
를 사용한 SDK 인증 AWS	5
AWS 액세스 포털 세션 시작	6
콘솔 로그인 자격 증명 사용	7
세부 인증 정보	7
Node.js 시작하기	8
시나리오	8
사전 조건	8
1단계: 패키지 구조 설정 및 클라이언트 패키지 설치	8
2단계: 필요한 가져오기 및 SDK 코드 추가	9
3단계: 예시 실행	11
브라우저에서 시작하기	12
시나리오	12
1단계: Amazon Cognito 자격 증명 풀 및 IAM 역할 생성	13
2단계: 생성된 IAM 역할에 정책 추가	13
3단계: Amazon S3 버킷 및 객체 추가	14
4단계: 브라우저 코드 설정	15
5단계: 예 실행	16
정리	16
React Native에서 시작하기	17
시나리오	17
사전 필수 작업	17
1단계: Amazon Cognito 자격 증명 풀 생성	18

2단계: 생성된 IAM 역할에 정책 추가	19
3단계: create-react-native-app을 사용하여 앱 생성	20
4단계: Amazon S3 패키지 및 기타 종속성 설치	20
5단계: React Native 코드 작성	21
6단계: 예제 실행	24
가능한 개선 사항	26
JavaScript용 SDK 설정	27
사전 조건	27
AWS Node.js 환경 설정	27
지원되는 웹 브라우저	28
SDK 설치	29
SDK 로드	30
SDK for JavaScript 구성	31
서비스별 구성	31
서비스별 구성 설정	32
AWS 리전 설정	32
클라이언트 클래스 생성자에서	32
환경 변수 사용	33
공유 구성 파일 사용	33
리전 설정을 위한 우선순위	33
자격 증명 설정	33
보안 인증에 대한 모범 사례	34
Node.js에서 자격 증명 설정	34
웹 브라우저에서 보안 인증 설정	38
Node.js 고려 사항	41
기본 제공 Node.js 모듈 사용	41
npm 패키지 사용	42
Node.js에서 maxSockets 구성	42
Node.js에서 연결 유지를 이용해 연결 재사용	43
Node.js용 프록시 구성	44
Node.js에서 인증서 번들 등록	45
브라우저 스크립트 고려 사항	45
브라우저용 SDK 빌드	45
교차 오리진 리소스 공유(CORS)	46
Webpack을 사용한 번들링	50
AWS 서비스 작업	54

서비스 객체 생성 및 직접 호출	54
서비스 객체 파라미터 지정	55
@smithy/types로 생성된 클라이언트	55
비동기식 서비스 직접 호출	58
비동기 직접 호출 관리	59
비동기/대기 사용	60
Promise 사용	61
콜백 함수 사용	62
서비스 클라이언트 요청 생성	63
서비스 클라이언트 응답 처리	64
응답에서 반환된 데이터 액세스	64
오류 정보 액세스	64
JSON 작업	64
서비스 객체 파라미터로서 JSON	65
AWS SDK for JavaScript 통화 로깅	66
미들웨어를 사용하여 요청 로깅	67
DynamoDB에서 AWS 계정 기반 엔드포인트 사용	67
Amazon S3 체크섬	68
객체 업로드	69
지침이 포함된 코드 예 하위 집합	71
JavaScript ES6/CommonJS 구문	72
AWS Elemental MediaConvert 예제	75
AWS Lambda 예제	91
Amazon Lex 예	91
Amazon Polly 예	92
Amazon Redshift 예	95
Amazon SES 예	103
Amazon SNS 예제	130
Amazon Transcribe 예	163
교차 서비스: Amazon EC2 인스턴스에서 Node.js 설정	174
교차 서비스: Amazon API Gateway 및 Lambda	177
교차 서비스: 예약된 Lambda 이벤트	191
교차 서비스: Amazon Lex 예	203
코드 예제	217
API Gateway	219
시나리오	219

Aurora	220
시나리오	219
Auto Scaling	221
작업	222
시나리오	219
Amazon Bedrock	264
시작하기	264
작업	222
Amazon Bedrock 런타임	268
시작하기	264
시나리오	219
Amazon Nova	282
Amazon Nova Canvas	299
Anthropic Claude	302
Cohere Command	312
Meta Llama	315
Mistral AI	322
Amazon Bedrock 에이전트	327
시작하기	264
작업	222
Amazon Bedrock Agents Runtime	341
작업	222
CloudWatch	346
작업	222
CloudWatch Events	355
작업	222
CloudWatch Logs	359
작업	222
시나리오	219
CodeBuild	375
작업	222
Amazon Cognito 자격 증명	379
시나리오	219
Amazon Cognito 자격 증명 공급자	380
시작하기	264
작업	222

시나리오	219
Amazon Comprehend	420
시나리오	219
Amazon DocumentDB	425
서버리스 예제	426
DynamoDB	427
시작하기	264
기본 사항	428
작업	222
시나리오	219
서버리스 예제	426
Amazon EC2	619
시작하기	264
기본 사항	428
작업	222
시나리오	219
Elastic Load Balancing - 버전 2	715
시작하기	264
작업	222
시나리오	219
AWS Entity Resolution	764
시작하기	264
작업	222
EventBridge	777
작업	222
시나리오	219
Amazon Glacier	783
작업	222
AWS Glue	785
시작하기	264
기본 사항	428
작업	222
HealthImaging	811
시작하기	264
작업	222
시나리오	219

IAM	873
시작하기	264
기본 사항	428
작업	222
시나리오	219
AWS IoT SiteWise	966
시작하기	264
기본 사항	428
작업	222
Kinesis	997
작업	222
서버리스 예제	426
Lambda	1004
시작하기	264
기본 사항	428
작업	222
시나리오	219
서버리스 예제	426
Amazon Lex	1059
시나리오	219
Amazon Location	1060
시작하기	264
기본 사항	428
작업	222
Amazon MSK	1091
서버리스 예제	426
Amazon Personalize	1093
작업	222
Amazon Personalize 이벤트	1110
작업	222
Amazon Personalize 런타임	1114
작업	222
Amazon Pinpoint	1118
작업	222
Amazon Polly	1123
시나리오	219

Amazon RDS	1127
시나리오	219
서버리스 예제	426
Amazon RDS 데이터 서비스	1132
시나리오	219
Amazon Redshift	1133
작업	222
Amazon Rekognition	1138
시나리오	219
Amazon S3	1140
시작하기	264
기본 사항	428
작업	222
시나리오	219
서버리스 예제	426
SageMaker AI	1272
시작하기	264
작업	222
시나리오	219
Secrets Manager	1311
작업	222
Amazon SES	1313
작업	222
시나리오	219
Amazon SNS	1338
시작하기	264
작업	222
시나리오	219
서버리스 예제	426
Amazon SQS	1379
시작하기	264
작업	222
시나리오	219
서버리스 예제	426
단계 함수	1410
작업	222

AWS STS	1412
작업	222
지원	1414
시작하기	264
기본 사항	428
작업	222
Systems Manager	1431
시작하기	264
기본 사항	428
작업	222
Amazon Textract	1459
시나리오	219
Amazon Transcribe	1464
작업	222
시나리오	219
Amazon Translate	1473
시나리오	219
보안	1479
데이터 보호	1479
자격 증명 및 액세스 관리	1480
대상	1481
ID를 통한 인증	1481
정책을 사용하여 액세스 관리	1483
IAM AWS 서비스 작업 방법	1484
AWS 자격 증명 및 액세스 문제 해결	1484
규정 준수 검증	1486
복원력	1486
인프라 보안	1487
최소 TLS 버전 적용	1488
Node.js에서 TLS 확인 및 적용	1488
브라우저 스크립트에서 TLS 확인 및 적용	1491
AWS SDK for JavaScript v3 요청에서 TLS 버전 검색	1492
v3로 마이그레이션	1493
codemod를 사용하여 v3로 마이그레이션	1493
codemod를 사용하여 기존 v2 코드 마이그레이션	1493
버전 3의 새 기능	1494

모듈화된 패키지	1495
코드 크기 비교	1496
v3에서 명령 직접 호출	1497
새 미들웨어 스택	1499
v2와 v3의 차이점은 무엇인가요?	1500
클라이언트 생성자	1500
보안 인증 공급자	1505
Amazon S3 고려 사항	1511
DynamoDB 문서 클라이언트	1513
웨이터 및 서명자	1514
특정 서비스 클라이언트에 대한 참고 사항	1516
보충 설명서	1519
문서 이력	1520
문서 기록	1520

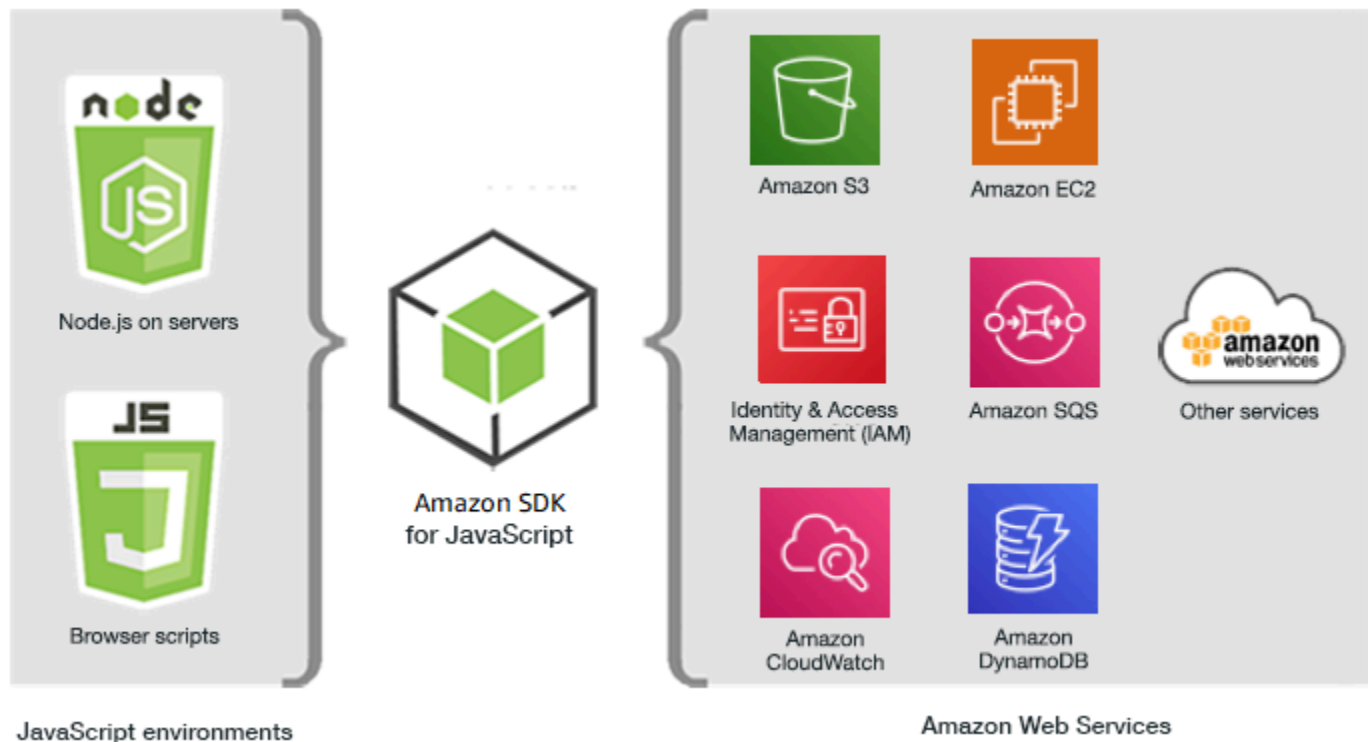
[AWS SDK for JavaScript V3 API 참조 안내서](#)는 AWS SDK for JavaScript 버전 3(V3)의 모든 API 작업을 자세히 설명합니다.

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 Guide](#)에서는 AWS 서비스용 JavaScript API를 제공합니다. JavaScript API를 사용하여 [Node.js](#)용 또는 브라우저용 라이브러리 또는 애플리케이션을 빌드할 수 있습니다.



SDK 시작하기

SDK를 직접 사용해 볼 준비가 되었다면 [시작하기](#)에 있는 예제를 따르세요.

개발 환경을 설정하려면 [JavaScript용 SDK 설정](#)을 참조하세요.

현재 SDK for JavaScript의 2.x 버전을 사용 중인 경우 구체적인 지침은 [v3로 마이그레이션](#)을 참조하세요.

AWS 서비스에 대한 코드 예제를 찾고 있다면 [SDK for JavaScript\(v3\) 코드 예](#) 섹션을 참조하세요.

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 안내서](#)에서 다음 내용을 참조하세요.

- [AWS SDK 및 도구 유지 관리 정책](#)
- [AWS SDK 및 도구 버전 지원 매트릭스](#)

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 Amplify에서 SDK 사용

브라우저 기반 웹, 모바일 및 하이브리드 앱의 경우 [GitHub의 AWS Amplify 라이브러리](#)를 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 참조하세요.

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 [Webpack으로 애플리케이션 번들링](#) 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 Repository](#)에서 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 re:Post](#)
- [AWS 코드 라이브러리의 JavaScript 예제](#)
- [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 시작하기

JavaScript용 AWS SDK는 브라우저 또는 Node.js 환경의 웹 서비스에 대한 액세스를 제공합니다. 이 섹션에서는 이러한 각 JavaScript 환경에서 AWS SDK for JavaScript로 작업하는 방법을 보여주는 연습을 시작했습니다.

주제

- [를 사용한 SDK 인증 AWS](#)
- [Node.js 시작하기](#)
- [브라우저에서 시작하기](#)
- [React Native에서 시작하기](#)

를 사용한 SDK 인증 AWS

를 사용하여 개발할 AWS 때 코드가 로 인증되는 방법을 설정해야 합니다 AWS 서비스. 환경 및 사용 가능한 액세스에 따라 다양한 방식으로 AWS 리소스에 대한 프로그래밍 방식 AWS 액세스를 구성할 수 있습니다.

인증 방법을 선택하고 SDK에 맞게 구성하려면 AWS SDK 및 도구 참조 안내서의 [Authentication and access](#)를 참조하세요.

로컬에서 개발 중이고 고용주로부터 설정 인증 방법을 받지 않은 신규 사용자에게 설정하는 것이 좋습니다 AWS IAM Identity Center. 이 방법에는 구성이 쉽고 AWS 액세스 포털 AWS CLI 에 정기적으로 로그인하기 위한 설치가 포함됩니다. 이 방법을 선택하는 경우 AWS SDK 및 도구 참조 안내서의 [IAM Identity Center authentication](#) 절차를 완료한 후 환경에 다음 요소가 포함되어야 합니다.

- 애플리케이션을 실행하기 전에 AWS 액세스 포털 세션을 시작하는 데 AWS CLI 사용하는 입니다.
- SDK에서 참조할 수 있는 구성 값 세트가 포함된 [default] 프로필이 있는 [shared AWSconfig file](#)입니다. 이 파일의 위치를 찾으려면 AWS SDK 및 도구 참조 가이드에서 [공유 파일의 위치](#)를 참조하세요.
- 공유 config 파일은 [region](#) 설정을 지정합니다. 이렇게 하면 SDK AWS 리전 가 AWS 요청에 사용하는 기본값이 설정됩니다. 이 리전은 사용할 리전이 지정되지 않은 SDK 서비스 요청에 사용됩니다.
- SDK는 AWS에 요청을 보내기 전에 프로필의 [SSO token provider configuration](#)을 사용하여 보안 인증을 얻습니다. IAM Identity Center 권한 세트에 연결된 IAM 역할인 sso_role_name 값은 애플리케이션에 AWS 서비스 사용되는데 대한 액세스를 허용합니다.

다음 샘플 config 파일은 SSO 토큰 공급자 구성으로 설정된 기본 프로필을 보여줍니다. 프로필의 `sso_session` 설정은 이름이 지정된 [sso-session section](#)을 참조합니다. `sso-session` 섹션에는 AWS 액세스 포털 세션을 시작하는 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 v3는 IAM Identity Center 인증을 사용하기 위해 애플리케이션에 추가 패키지(예: SSO 및 SS00IDC)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

에 액세스하는 애플리케이션을 실행하기 전에 SDK가 IAM Identity Center 인증을 사용하여 자격 증명을 확인하려면 활성 AWS 액세스 포털 세션이 AWS 서비스필요합니다. 구성된 세션 길이에 따라 결국 액세스가 만료되고 SDK에 인증 오류가 발생합니다. AWS 액세스 포털에 로그인하려면에서 다음 명령을 실행합니다 AWS CLI.

```
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.

활성 세션이 이미 있는지 선택적으로 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 있고를 실행하는 경우 `aws sso login` 자격 증명을 제공할 필요가 없습니다.

로그인 프로세스에서 데이터에 대한 AWS CLI 액세스를 허용하라는 메시지가 표시될 수 있습니다. AWS CLI 는 SDK for Python을 기반으로 구축되므로 권한 메시지에 `botocore` 이름의 변형이 포함될 수 있습니다.

콘솔 로그인 자격 증명 사용

기존 AWS Management Console 로그인 자격 증명을 사용하여 AWS 서비스에 프로그래밍 방식으로 액세스할 수 있습니다. 브라우저 기반 인증 흐름 후는 AWS CLI 및 AWS SDK for JavaScript와 같은 로컬 개발 도구에서 작동하는 임시 자격 증명을 AWS 생성합니다. 이 기능은 AWS CLI 자격 증명을 구성하고 관리하는 프로세스를 간소화합니다. 시작하는 방법을 알아보려면 [콘솔 자격 증명을 사용하여 AWS 로컬 개발에 로그인](#) 지침을 따르세요.

`aws login` 명령을 실행하면 활성 콘솔 세션에서 선택하거나 브라우저 기반 인증 흐름을 통해 로그인할 수 있으며, 그러면 임시 자격 증명이 자동으로 생성됩니다. AWS SDK for JavaScript는 만료 5분 전에 자격 증명을 자동으로 새로 고치며 각 자격 증명 세트는 최대 12시간 동안 유효합니다. 자세한 내용은 [fromLoginCredentials\(\)](#)을 참조하세요.

세부 인증 정보

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

에 액세스할 때 임시 자격 증명을 사용합니다 AWS. 인간 사용자의 자격 증명 공급자를 사용하여 임시 자격 증명을 제공하는 역할을 수입하여 AWS 계정에 대한 페더레이션 액세스를 제공할 수 있습니다. 중앙 액세스 관리를 위해 AWS IAM Identity Center (IAM Identity Center)을 사용하여 계정에 대한 액세스 권한과 해당 계정 내 권한을 관리하는 것이 좋습니다. 더 많은 대안을 보려면 다음을 참조하세요.

- 모범 사례에 대해 자세히 알아보려면 IAM 사용 설명서에서 [IAM의 보안 모범 사례](#)를 참조하세요.
- 단기 AWS 자격 증명을 생성하려면 IAM 사용 설명서의 [임시 보안 자격 증명을 참조하세요](#).

- 다른 AWS SDK for JavaScript V3 자격 증명 공급자에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [표준화된 자격 증명 공급자](#)를 참조하세요. AWS SDKs

Node.js 시작하기

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

시나리오

다음은 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 [를 사용한 SDK 인증 AWS](#) 단원을 참조하십시오.
- 개발을 위해 [Node.js](#)의 활성 LTS 버전을 사용하여 Node.js. AWS recommends를 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.js`라는 파일에 다음 코드를 추가합니다.

```
// This is used for getting user input.
import { createInterface } from "node: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 "node: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 명령을 사용하여 AWS CLI `aws sso login` 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 섹션에서는 브라우저에서 AWS SDK for JavaScript 버전 3(V3)을 실행하는 방법을 보여주는 예제를 안내합니다.

Note

브라우저에서 V3를 실행하는 것은 버전 2(V2)와 약간 다릅니다. 자세한 내용은 [V3에서 브라우저 사용](#) 단원을 참조하십시오.

AWS SDK for JavaScript의 (V3)를 사용하는 다른 예는 섹션을 참조하세요 [SDK for JavaScript\(v3\) 코드 예](#).

이 웹 애플리케이션 예는 다음을 보여줍니다.

- 인증을 위해 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 예제에서는 인증 AWS IAM Identity Center 예를 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 설명서](#)를 참조하세요.

이 예는 Amazon S3 버킷에서 읽을 IAM 역할을 맡는 웹 앱을 설정하고 실행하는 방법을 보여줍니다. 이 예에서는 React 프론트엔드 라이브러리와 Vite 프론트엔드 도구를 사용하여 JavaScript 개발 환경

을 제공합니다. 웹 앱은 Amazon Cognito 자격 증명 풀을 사용하여 AWS 서비스에 액세스하는 데 필요한 자격 증명을 제공합니다. 포함된 코드 예제는 웹 앱에서 AWS SDK for JavaScript를 로드하고 사용하기 위한 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 인증되지 않은 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. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/cognito/> 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 설정](#)에서 **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. 예 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/iam/> IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 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 모든 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. 예 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/s3/> 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.

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

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

- 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

- [Node.js](#)를 설치합니다.
- 명령줄에서 [AWS 코드 예 리포지토리](#)를 복제합니다.

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

- 다음과 같이 애플리케이션 예로 이동합니다.

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

- 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 생성](#)에서 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 추가](#)에서 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 콘솔](#)에서, 생성한 객체와 버킷(예: `getstartedbucket`)을 삭제합니다.

- [IAM 콘솔](#)에서, 역할 이름(예: getStartedRole)을 삭제합니다.
- [Amazon Cognito 콘솔](#)에서, 자격 증명 풀 이름(예: getStartedPool)을 삭제합니다.

React Native에서 시작하기

이 자습서에서는 [React Native CLI](#)를 사용하여 React Native 앱을 생성하는 방법을 보여줍니다.



이 자습서에서는 다음을 보여줍니다.

- 프로젝트에서 사용하는 AWS SDK for JavaScript 버전 3(V3) 모듈을 설치하고 포함하는 방법.
- Amazon Simple Storage Service(Amazon S3)에 연결하여 Amazon S3 버킷을 생성하고 삭제하는 코드를 작성하는 방법입니다.

시나리오

Amazon S3는 클라우드 서비스로, 언제 어디서나 웹에 원하는 양의 데이터를 저장하고 검색할 수 있습니다. React Native는 모바일 애플리케이션을 생성할 수 있는 개발 프레임워크입니다. 이 자습서에서는 Amazon S3에 연결하여 Amazon S3 버킷을 생성 및 삭제할 수 있는 React Native 앱을 생성하는 방법을 보여줍니다.

앱은 다음 AWS SDK for JavaScript APIs를 사용합니다.

- [CognitoIdentityClient](#) 생성자
- [S3](#) 생성자

사전 필수 작업

Note

다른 자습서나 기존 구성을 통해 다음 단계를 이미 완료한 경우 해당 단계를 건너뛰십시오.

이 단원에서는 이 자습서를 완료하는 데 필요한 최소 설정을 제공합니다. 이 내용을 전체 설정으로 간주해서는 안 됩니다. 자세한 내용은 [JavaScript용 SDK 설정](#) 단원을 참조하십시오.

- 다음 도구를 설치합니다.
 - [npm](#)
 - [Node.js](#)
 - iOS에서 테스트하는 경우 [Xcode](#)
 - Android에서 테스트하는 경우 [Android Studio](#)
- [React Native 개발 환경](#) 설정
- 이러한 노드 TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 및 타사 모듈을 설치합니다. [GitHub](#)의 지침을 따릅니다.
- AWS 서비스를 사용하여 개발할 AWS 때 코드가 인증하는 방법을 설정해야 합니다. 자세한 내용은 [클라우드 사용 SDK 인증 AWS](#) 단원을 참조하십시오.

Note

이 예제에서 IAM 역할은 AmazonS3FullAccess 권한을 사용하도록 설정해야 합니다.

1단계: Amazon Cognito 자격 증명 풀 생성

이 연습에서는 Amazon Cognito ID 풀을 생성해서 사용하여 Amazon S3 서비스용 앱에 대한 미인증 액세스 권한을 제공합니다. 자격 증명 풀을 생성하면 두 개의 AWS Identity and Access Management (IAM) 역할도 생성됩니다. 하나는 자격 증명 공급자가 인증한 사용자를 지원하는 역할이고 다른 하나는 인증되지 않은 게스트 사용자를 지원하는 역할입니다.

이 연습에서는 인증되지 않은 사용자 역할만 사용해 작업하여 작업에 집중할 수 있도록 합니다. 자격 증명 공급자 및 인증된 사용자에 대한 지원은 나중에 통합할 수 있습니다.

Amazon Cognito ID 풀을 생성하려면

1. 에 로그인 AWS Management Console 하고 Amazon Web Services 콘솔에서 Amazon Cognito [콘솔](#)을 엽니다.
2. 콘솔 열기 페이지에서 ID 풀을 선택합니다.
3. 다음 페이지에서 Create new identity pool(새 자격 증명 풀 만들기)을 선택합니다.

 Note

다른 ID 풀이 없는 경우 Amazon Cognito 콘솔에서는 이 페이지를 건너뛰고 대신 다음 페이지를 엽니다.

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: `getStartedReactRole`)을 입력합니다.
6. 속성 구성에서 ID 풀 이름에 이름(예: `getStartedReactPool`)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 ID 풀의 ID 풀 ID 및 리전을 기록해 둡니다. 이러한 값은 브라우저 스크립트에서 `##` 및 `identityPoolId`를 바꾸는 데 필요합니다.

Amazon Cognito ID 풀을 생성하면 React Native 앱에 필요한 Amazon S3에 대한 권한을 추가할 준비가 된 것입니다.

2단계: 생성된 IAM 역할에 정책 추가

Amazon S3에 대한 브라우저 스크립트 액세스를 활성화하여 Amazon S3 버킷을 생성하고 삭제하려면 Amazon Cognito ID 풀에 대해 생성된 인증되지 않은 IAM 역할을 사용합니다. 이 단계를 수행하려면 해당 역할에 IAM 정책을 추가해야 합니다. IAM 역할에 대한 자세한 내용은 IAM 사용 설명서의 [AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요](#).

미인증 사용자와 연결된 IAM 역할에 Amazon S3 정책을 추가하는 방법

1. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/iam/> IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 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 모든 AWS 서비스에 대한 액세스를 활성화할 수 있습니다.

6. 권한 추가를 선택합니다.

Amazon Cognito ID 풀을 생성한 후 미인증 사용자의 IAM 역할에 Amazon S3에 대한 권한을 추가하면 앱을 빌드할 준비가 된 것입니다.

3단계: create-react-native-app을 사용하여 앱 생성

다음 명령을 실행하여 React Native 앱을 생성합니다.

```
npx react-native init ReactNativeApp --npm
```

4단계: Amazon S3 패키지 및 기타 종속성 설치

프로젝트의 디렉터리 내에서 다음 명령을 실행하여 Amazon S3 패키지를 설치합니다.

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

이 명령은 프로젝트에 Amazon S3 패키지를 설치하고 `package.json`을 업데이트하여 Amazon S3를 프로젝트 종속성으로 나열합니다. <https://www.npmjs.com/npm> 웹 사이트에서 '@aws-sdk'를 검색하여 이 패키지에 대한 정보를 찾을 수 있습니다.

이러한 패키지와 연결된 코드는 프로젝트의 `node_modules` 하위 디렉터리에 설치됩니다.

Node.js 패키지 설치에 대한 자세한 내용은 [로컬에서 패키지 다운로드 및 설치](#) 및 [npm\(Node.js 패키지 관리자\) 웹 사이트의 Node.js 모듈 생성](#) 섹션을 참조하세요. AWS SDK for JavaScript 다운로드 및 설치에 대한 자세한 내용은 [섹션을 참조하세요](#) [SDK for JavaScript 설치](#).

인증에 필요한 다른 종속성을 설치합니다.

```
npm install @aws-sdk/client-cognito-identity @aws-sdk/credential-provider-cognito-identity
```


5단계: React Native 코드 작성

다음 코드를 App.tsx에 추가합니다. *identityPoolId* 및 ##을 Amazon S3 버킷이 생성될 ID 풀 ID 및 리전으로 바꿉니다.

```
import React, { useCallback, useState } from "react";
import { Button, StyleSheet, Text, TextInput, View } from "react-native";
import "react-native-get-random-values";
import "react-native-url-polyfill/auto";

import {
  S3Client,
  CreateBucketCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";

const client = new S3Client({
  // The AWS Region where the Amazon Simple Storage Service (Amazon S3) bucket will be
  // created. Replace this with your Region.
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    // Replace the value of 'identityPoolId' with the ID of an Amazon Cognito identity
    // pool in your Amazon Cognito Region.
    identityPoolId: "us-east-1:edbe2c04-7f5d-469b-85e5-98096bd75492",
    // Replace the value of 'region' with your Amazon Cognito Region.
    clientConfig: { region: "us-east-1" },
  }),
});

enum MessageType {
  SUCCESS = 0,
  FAILURE = 1,
  EMPTY = 2,
}

const App = () => {
  const [bucketName, setBucketName] = useState("");
  const [msg, setMsg] = useState<{ message: string; type: MessageType }>({
    message: "",
    type: MessageType.EMPTY,
  });
}
```

```

const createBucket = useCallback(async () => {
  setMsg({ message: "", type: MessageType.EMPTY });

  try {
    await client.send(new CreateBucketCommand({ Bucket: bucketName }));
    setMsg({
      message: `Bucket "${bucketName}" created.`,
      type: MessageType.SUCCESS,
    });
  } catch (e) {
    console.error(e);
    setMsg({
      message: e instanceof Error ? e.message : "Unknown error",
      type: MessageType.FAILURE,
    });
  }
}, [bucketName]);

const deleteBucket = useCallback(async () => {
  setMsg({ message: "", type: MessageType.EMPTY });

  try {
    await client.send(new DeleteBucketCommand({ Bucket: bucketName }));
    setMsg({
      message: `Bucket "${bucketName}" deleted.`,
      type: MessageType.SUCCESS,
    });
  } catch (e) {
    setMsg({
      message: e instanceof Error ? e.message : "Unknown error",
      type: MessageType.FAILURE,
    });
  }
}, [bucketName]);

return (
  <View style={styles.container}>
    {msg.type !== MessageType.EMPTY && (
      <Text
        style={
          msg.type === MessageType.SUCCESS
            ? styles.successText
            : styles.failureText
        }
      />
    )}
  </View>
)

```

```

    >
      {msg.message}
    </Text>
  })
</View>
  <TextInput
    onChangeText={({text}) => setBucketName(text)}
    autoCapitalize={"none"}
    value={bucketName}
    placeholder={"Enter Bucket Name"}
  />
  <Button color="#68a0cf" title="Create Bucket" onPress={createBucket} />
  <Button color="#68a0cf" title="Delete Bucket" onPress={deleteBucket} />
</View>
</View>
);
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: "center",
    justifyContent: "center",
  },
  successText: {
    color: "green",
  },
  failureText: {
    color: "red",
  },
});

export default App;

```

이 코드는 먼저 필요한 React, React Native 및 AWS SDK 종속성을 가져옵니다.

함수 앱 내부:

- S3Client 객체가 생성되어 이전에 생성한 Amazon Cognito ID 풀을 사용하여 자격 증명을 지정합니다.
- 메서드 createBucket 및 deleteBucket은 각각 지정된 버킷을 생성하고 삭제합니다.

- React Native View에는 사용자가 Amazon S3 버킷 이름을 지정할 수 있는 텍스트 입력 필드와 지정된 Amazon S3 버킷을 생성하고 삭제할 수 있는 버튼이 표시됩니다.

전체 JavaScript 페이지는 [GitHub](#)에서 확인할 수 있습니다.

6단계: 예제 실행

Note

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

예제를 실행하려면 npm을 사용하여 `web`, `ios` 또는 `android` 명령을 실행합니다.

다음은 macOS에서 `ios` 명령을 실행하는 출력의 예입니다.

```
$ npm run ios

> ReactNativeApp@0.0.1 ios /Users/trivikr/workspace/ReactNativeApp
> react-native run-ios

info Found Xcode workspace "ReactNativeApp.xcworkspace"
info Launching iPhone 11 (iOS 14.2)
info Building (using "xcodebuild -workspace ReactNativeApp.xcworkspace -configuration
  Debug -scheme ReactNativeApp -destination id=706C1A97-FA38-407D-AD77-CB4FCA9134E9")
success Successfully built the app
info Installing "/Users/trivikr/Library/Developer/Xcode/DerivedData/ReactNativeApp-
cfhmsyhptwflqqejyspdqgjestr/Build/Products/Debug-iphonesimulator/ReactNativeApp.app"
info Launching "org.reactjs.native.example.ReactNativeApp"

success Successfully launched the app on the simulator
```

다음은 macOS에서 `android` 명령을 실행하는 출력의 예입니다.

```
$ npm run android

> ReactNativeApp@0.0.1 android
> react-native run-android
```

```

info Running jetifier to migrate libraries to AndroidX. You can disable it using "--no-jetifier" flag.
Jetifier found 970 file(s) to forward-jetify. Using 12 workers...
info Starting JS server...
info Launching emulator...
info Successfully launched emulator.
info Installing the app...

> Task :app:stripDebugDebugSymbols UP-TO-DATE
Compatible side by side NDK version was not found.

> Task :app:installDebug
02:18:38 V/ddms: execute: running am get-config
02:18:38 V/ddms: execute 'am get-config' on 'emulator-5554' : EOF hit. Read: -1
02:18:38 V/ddms: execute: returning
Installing APK 'app-debug.apk' on 'Pixel_3a_API_30_x86(AVD) - 11' for app:debug
02:18:38 D/app-debug.apk: Uploading app-debug.apk onto device 'emulator-5554'
02:18:38 D/Device: Uploading file onto device 'emulator-5554'
02:18:38 D/ddms: Reading file permission of /Users/trivikr/workspace/ReactNativeApp/android/app/build/outputs/apk/debug/app-debug.apk as: rw-r--r--
02:18:40 V/ddms: execute: running pm install -r -t "/data/local/tmp/app-debug.apk"
02:18:41 V/ddms: execute 'pm install -r -t "/data/local/tmp/app-debug.apk"' on
'emulator-5554' : EOF hit. Read: -1
02:18:41 V/ddms: execute: returning
02:18:41 V/ddms: execute: running rm "/data/local/tmp/app-debug.apk"
02:18:41 V/ddms: execute 'rm "/data/local/tmp/app-debug.apk"' on 'emulator-5554' : EOF
hit. Read: -1
02:18:41 V/ddms: execute: returning
Installed on 1 device.

Deprecated Gradle features were used in this build, making it incompatible with Gradle
7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.2/userguide/
command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 6s
27 actionable tasks: 2 executed, 25 up-to-date
info Connecting to the development server...
8081
info Starting the app on "emulator-5554"...
Starting: Intent { cmp=com.reactnativeapp.MainActivity }

```

생성하거나 삭제할 버킷 이름을 입력하고 버킷 생성 또는 버킷 삭제를 클릭합니다. 해당 명령이 Amazon S3로 전송되고 성공 또는 오류 메시지가 표시됩니다.

Success: Bucket "test-bucket-name-123" created.

test-bucket-name-123

Create Bucket

Delete Bucket

가능한 개선 사항

다음은 React Native 앱에서 AWS SDK for JavaScript를 사용하여 추가로 탐색하는 데 사용할 수 있는 이 애플리케이션의 변형입니다.

- Amazon S3 버킷을 나열하는 버튼을 추가하고 나열된 각 버킷 옆에 삭제 버튼을 제공합니다.
- 버튼을 추가하여 텍스트 객체를 버킷에 넣습니다.
- 인증된 IAM 역할과 함께 사용할 수 있도록 Facebook 또는 Amazon 등과 같은 ID 공급자를 통합합니다.

JavaScript용 SDK 설정

이 단원의 항목에서는 SDK에서 지원하는 웹 서비스에 액세스할 수 있도록 SDK for JavaScript를 설치하고 로드하는 방법을 설명합니다.

주제

- [사전 조건](#)
- [SDK for JavaScript 설치](#)
- [SDK for JavaScript 로드](#)

사전 조건

개발을 위해 [Node.js의 활성 LTS 버전을 사용하여 Node.js.recommends를 설치합니다](#). AWS

주제

- [AWS Node.js 환경 설정](#)
- [지원되는 웹 브라우저](#)

AWS Node.js 환경 설정

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

- 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. AWS Marketplace에서 Node.js를 검색하고 사전 설치된 Node.js 버전(32비트 또는 64비트)이 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](#) 단원을 참조하세요.
- 를 사용하여 서버리스 환경을 생성 AWS Lambda 하여 Node.js를 Lambda 함수로 실행합니다. Lambda 함수 내에서 Node.js를 사용하는 방법에 관한 자세한 내용은 AWS Lambda 개발자 안내서의 [프로그래밍 모델\(Node.js\)](#) 단원을 참조하세요.
- Node.js 애플리케이션을 배포합니다 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.

지원되는 웹 브라우저

는 모든 최신 웹 브라우저를 AWS SDK for JavaScript 지원합니다.

SDK for JavaScript는 버전 3.567.0 이상에서 다음과 같은 최소 버전을 지원하는 ES2021 아티팩트를 사용합니다.

브라우저	버전
Google Chrome	85.0+
Mozilla Firefox	80.0+
Opera	71.0+
Microsoft Edge	85.0+
Apple Safari	14.1+
삼성 인터넷	14.0+

SDK for JavaScript는 버전 3.183.0~3.566.0에서 다음과 같은 최소 버전을 지원하는 ES2020 아티팩트를 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+

SDK for JavaScript는 버전 3.182.0 이하에서 다음과 같은 최소 버전을 지원하는 ES5 아티팩트를 사용합니다.

브라우저	버전
Google Chrome	49.0+
Mozilla Firefox	45.0+
Opera	36.0+
Microsoft Edge	12.0+
Windows Internet Explorer	해당 사항 없음
Apple Safari	9.0+
Android 브라우저	76.0+
UC 브라우저	12.12+
삼성 인터넷	5.0+

Note

와 같은 프레임워크는 JavaScript용 SDK와 동일한 브라우저 지원을 제공하지 않을 AWS Amplify 수 있습니다. 자세한 내용은 [AWS Amplify 설명서](#)를 참조하세요.

SDK for JavaScript 설치

일부 서비스는 SDK 또는 일부 AWS 리전에서 즉시 사용할 수 없습니다.

[Node.js 패키지 관리자인 npm](#)을 AWS SDK for JavaScript 사용하여 서비스 설치하려면 명령 프롬프트에 다음 명령을 입력합니다. 여기서 **SERVICE**는와 같은 서비스의 이름입니다s3.

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

AWS SDK for JavaScript 서비스 클라이언트 패키지의 전체 목록은 [AWS SDK for JavaScript API 참조 가이드](#)를 참조하세요.

SDK for JavaScript 로드

SDK를 설치한 후 `import`를 사용하여 노드 애플리케이션에 클라이언트 패키지를 로드할 수 있습니다. 예를 들어 Amazon S3 클라이언트와 Amazon S3 [ListBuckets](#) 명령을 로드하려면 다음을 사용합니다.

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

SDK for JavaScript 구성

API를 사용하여 웹 서비스를 간접적으로 호출하기 위해 SDK for JavaScript를 사용하기 전에 SDK를 구성해야 합니다. 최소한 다음을 구성해야 합니다.

- 서비스를 요청할 AWS 리전
- 를 사용하여 코드를 인증하는 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 SDKs 및 도구 참조 가이드](#)에는 많은 AWS SDKs.

이 단원의 주제에서는 웹 브라우저에서 실행되는 JavaScript 및 Node.js에 대해 SDK for JavaScript를 구성하는 방법을 설명합니다.

주제

- [서비스별 구성](#)
- [AWS 리전 설정](#)
- [자격 증명 설정](#)
- [Node.js 고려 사항](#)
- [브라우저 스크립트 고려 사항](#)

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 생성자에게 전달할 수 있습니다. 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

SDK for JavaScript에 사용하는 각 서비스에는 해당 서비스에 대한 API의 일부인 서비스 객체를 통해 액세스합니다. 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 리전의 Amazon EC2 객체에 액세스해야 하는 경우 각 리전에 대해 Amazon EC2 서비스 객체를 생성한 다음 그에 따라 각 서비스 객체의 리전 구성을 설정합니다.

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

AWS 리전 설정

AWS 리전은 동일한 지리적 영역에 있는 명명된 AWS 리소스 집합입니다. 리전의 한 가지 예로 미국 동부(버지니아 북부) 리전인 `us-east-1`을 들 수 있습니다. SDK가 해당 리전의 서비스에 액세스할 수 있도록 SDK for JavaScript에서 서비스 클라이언트를 생성할 때 리전을 지정합니다. 일부 서비스는 특정 리전에서만 사용할 수 있습니다.

SDK for JavaScript는 기본적으로 리전을 선택하지 않습니다. 그러나 환경 변수 또는 공유 구성 `config` 파일을 사용하여 AWS 리전을 설정할 수 있습니다.

클라이언트 클래스 생성자에서

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

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

환경 변수 사용

AWS_REGION 환경 변수를 사용하여 리전을 설정할 수 있습니다. 이 변수를 정의하면 SDK for JavaScript가 해당 변수를 읽고 사용합니다.

공유 구성 파일 사용

공유 자격 증명 파일을 사용하면 SDK에서 사용할 자격 증명을 저장할 수 있는 것과 마찬가지로 AWS 리전 및 기타 구성 설정을 config SDK에서 사용할 공유 파일에 보관할 수 있습니다. AWS_SDK_LOAD_CONFIG 환경 변수가 진리 값(truthy value)으로 설정된 경우 SDK for JavaScript는 로드 시 config 파일을 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 는 자격 증명을 사용하여 서비스를 호출하는 사람과 요청된 리소스에 대한 액세스가 허용되는지 여부를 식별합니다.

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

웹 브라우저에서 Node.js와 JavaScript 간에 서로 다른 인증 자격 증명을 설정하는 방법에는 여러 가지가 있습니다. 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.js에서 자격 증명 설정](#)
- [웹 브라우저에서 보안 인증 설정](#)

Node.js에서 자격 증명 설정

로컬에서 개발 중이고 고용주로부터 설정 인증 방법을 받지 않은 신규 사용자에게 설정하는 것이 좋습니다 AWS IAM Identity Center. 자세한 내용은 [를 사용한 SDK 인증 AWS](#) 단원을 참조하십시오.

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 V3는 Node.js에 기본 자격 증명 공급자 체인을 제공하므로 자격 증명 공급자를 명시적으로 제공할 필요가 없습니다. 기본 [보안 인증 공급자 체인](#)은 보안 인증이 소스 중 하나에서

반환될 때까지 지정된 우선순위에 따라 여러 가지 다양한 소스의 보안 인증을 확인하려고 시도합니다. SDK for JavaScript V3의 보안 인증 공급자 체인은 [여기에서](#) 찾을 수 있습니다.

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 직접 값을 설정하는 것이 항상 우선하며, 환경 변수로 설정한 다음 공유 AWS config 파일에서 설정합니다. 자세한 내용은 AWS SDK 및 도구 참조 안내서의 [Precedence of settings](#)를 참조하세요.

AWS SDKs 및 도구 참조 가이드에는 모든 AWS SDKs 및에서 사용하는 SDK 구성 설정에 대한 정보가 있습니다. AWS CLI, 공유 AWS config 파일을 통해 SDK를 구성하는 방법에 대한 자세한 내용은 [공유 구성 및 자격 증명 파일을 참조](#)하세요. 환경 변수 설정을 통해 SDK를 구성하는 방법에 대해 자세히 알아보려면 [Environment variables support](#) 단원을 참조하세요.

를 인증하기 위해 AWS는 다음 표에 나열된 순서대로 자격 증명 공급자를 AWS SDK for JavaScript 확인합니다.

우선 순위별 AWS SDK for JavaScript API 참조 자격 증명 공급자 메서드	사용 가능한 보안 인증 공급자	AWS SDKs 및 도구 참조 가이드
fromEnv()	AWS 환경 변수의 액세스 키	AWS 액세스 키
fromSSO()	AWS IAM Identity Center. 이 설명서의 를 사용한 SDK 인증 AWS 단원을 참조하세요.	IAM Identity Center 보안 인증 공급자
fromIni()	AWS 공유 config 및 credentials 파일에서 키 액세스	AWS 액세스 키
	신뢰할 수 있는 엔터티 공급자 (예: AWS_ROLE_ARN)	IAM 역할 수임

우선 순위별 AWS SDK for JavaScript API 참조 자격 증명 공급자 메서드	사용 가능한 보안 인증 공급자	AWS SDKs 및 도구 참조 가이드
	의 웹 자격 증명 토큰 AWS Security Token Service (AWS STS)	웹 자격 증명 또는 OpenID Connect와 페더레이션
	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	IAM Identity Center 보안 인증 공급자
	로그인 자격 증명 공급자	로그인 자격 증명 공급자
fromLoginCredentials()	로그인 자격 증명 공급자	로그인 자격 증명 공급자
fromProcess()	프로세스 보안 인증 공급자	프로세스 보안 인증 공급자
fromTokenFile()	의 웹 자격 증명 토큰 AWS Security Token Service (AWS STS)	웹 자격 증명 또는 OpenID Connect와 페더레이션
fromContainerMetadata()	Amazon Elastic Container Service(Amazon ECS) 보안 인증	컨테이너 보안 인증 공급자

우선 순위별 AWS SDK for JavaScript API 참조 자격 증명 공급자 메서드	사용 가능한 보안 인증 공급자	AWS SDKs 및 도구 참조 가이드
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 [를 사용한 SDK 인증 AWS](#) 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.js의 자격 증명 로드](#)
- [Node.js Lambda 함수의 자격 증명 로드](#)

Amazon EC2의 IAM 역할에서 Node.js의 자격 증명 로드

Amazon EC2 인스턴스에서 Node.js 애플리케이션을 실행하는 경우 Amazon EC2의 IAM 역할을 활용하여 해당 인스턴스에 보안 인증을 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 함수의 자격 증명 로드

AWS Lambda 함수를 생성할 때 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 보안 인증 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

스크립트에서 AWS 보안 인증을 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 사용자 인증](#)

Amazon Cognito 자격 증명을 사용하여 사용자 인증

브라우저 스크립트에 대한 AWS 보안 인증을 얻는 권장 방법은 Amazon Cognito 자격 증명 보안 인증 클라이언트 `CognitoIdentityClient`를 사용하는 것입니다. 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 STS)에서 웹 자격 증명 연동 지원을 사용하여 인증 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 보안 인증 객체 구성

아직 자격 증명 풀을 생성하지 않은 경우 Amazon Cognito 클라이언트를 구성하기 전에 [Amazon Cognito 콘솔](#)에서 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-providers에서 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 = 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 코드는 JavaScript이지만 Node.js AWS SDK for JavaScript 에서 사용하는 것은 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.

Note

AWS에서는 개발을 위해 Node.js의 활성 LTS 버전을 사용할 것을 권장합니다.

기본 제공 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 documentation](#)을 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.com>을 참조하세요. [GitHub](#)에서 사용할 수 있는 추가 Node.js 패키지에 대해 알아볼 수도 있습니다.

Node.js에서 maxSockets 구성

Node.js에서 오리진당 최대 연결 수를 설정할 수 있습니다. maxSockets이 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 maxSockets를 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
  });
});
```

값 또는 Agent 객체를 제공하지 않는 경우 SDK for JavaScript는 50의 maxSockets 값을 사용합니다. Agent 객체를 제공하면 해당 maxSockets 값이 사용됩니다. Node.js에서 maxSockets 설정에 대한 자세한 내용은 [Node.js 설명서](#)를 참조하세요.

의 v3.521.0부터 다음 [간편 구문을 사용하여](#)를 구성할 AWS SDK for JavaScript 수 있습니다 requestHandler.

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

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

Node.js에서 연결 유지를 이용해 연결 재사용

기본 Node.js HTTP/HTTPS 에이전트는 모든 새 요청에 대해 새로운 TCP 연결을 생성합니다. 새 연결 설정 비용을 방지하기 위해서는 기본적으로 TCP 연결을 AWS SDK for JavaScript 재사용합니다.

Amazon DynamoDB 쿼리와 같은 수명이 짧은 작업의 경우 TCP 연결 설정에 따른 지연 시간 오버헤드가 작업 자체보다 클 수 있습니다. 또한 저장 시 DynamoDB 암호화와 통합되므로 데이터베이스에서 각 작업에 대해 새 AWS KMS 캐시 항목을 다시 설정해야 하는 지연 시간이 [AWS KMS](#) 발생할 수 있습니다. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/encryption.howitworks.html>

TCP 연결을 재사용하지 않으려면, 다음 DynamoDB 클라이언트 예시에서와 같이 서비스별 클라이언트 기준으로 keepAlive를 사용하여 활성 상태의 연결 재사용을 비활성화할 수 있습니다.

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

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

keepAlive가 활성화된 경우 기본값인 1000ms인 keepAliveMsecs를 사용하여 TCP 연결 유지 패킷에 대한 초기 지연을 설정할 수도 있습니다. 자세한 내용은 [Node.js 설명서](#)를 참조하세요.

Node.js용 프록시 구성

인터넷에 직접 연결할 수 없는 경우 SDK for JavaScript는 타사 HTTP 에이전트를 통해 HTTP 또는 HTTPS 프록시 사용을 지원합니다.

타사 HTTP 에이전트를 찾으려면 [npm](#)에서 "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

httpAgent는 httpsAgent와 동일하지 않으며 클라이언트에서 대부분의 직접 호출이 https로 이루어지므로 둘 다 설정해야 합니다.

Node.js에서 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 주제에서는 브라우저 스크립트 AWS SDK for JavaScript 에서 사용하기 위한 특별한 고려 사항을 설명합니다.

주제

- [브라우저용 SDK 빌드](#)
- [교차 오리진 리소스 공유\(CORS\)](#)
- [Webpack으로 애플리케이션 번들링](#)

브라우저용 SDK 빌드

SDK for JavaScript 버전 2(V2)와 달리 V3는 기본 서비스 집합에 대한 지원이 포함된 JavaScript 파일로 제공됩니다. 대신 V3를 사용하면 필요한 SDK for JavaScript 파일만 번들로 제공하고 브라우저에 포

함할 수 있으므로 오버헤드가 줄어듭니다. Webpack을 사용하여 필수 SDK for JavaScript 파일과 필요한 추가 타사 패키지를 단일 Javascript 파일로 번들링하고 <script> 태그를 사용하여 브라우저 스크립트에 이 파일을 로드하는 것이 좋습니다. Webpack에 관한 자세한 내용은 [Webpack으로 애플리케이션 번들링](#) 단원을 참조하세요.

브라우저에서 CORS를 적용하는 환경 외부에서 SDK를 사용하며 SDK for JavaScript에서 제공하는 모든 서비스에 액세스하려는 경우 리포지토리를 복제하고 기본 호스팅 버전의 SDK를 빌드하는 동일한 빌드 도구를 실행하여 SDK의 사용자 지정 사본을 로컬에서 빌드할 수 있습니다. 다음 섹션에서는 추가 서비스 및 API 버전으로 SDK를 빌드하는 단계를 설명합니다.

SDK 빌더를 사용하여 SDK for JavaScript 빌드

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 [Webpack으로 애플리케이션 번들링](#) 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 JavaScript를 사용해 브라우저 애플리케이션을 개발하는 경우 CORS는 중요한 고려 대상입니다. JavaScript 환경에서 CORS 보안을 적용하는 경우 이 서비스와 함께 CORS를 구성해야 합니다.

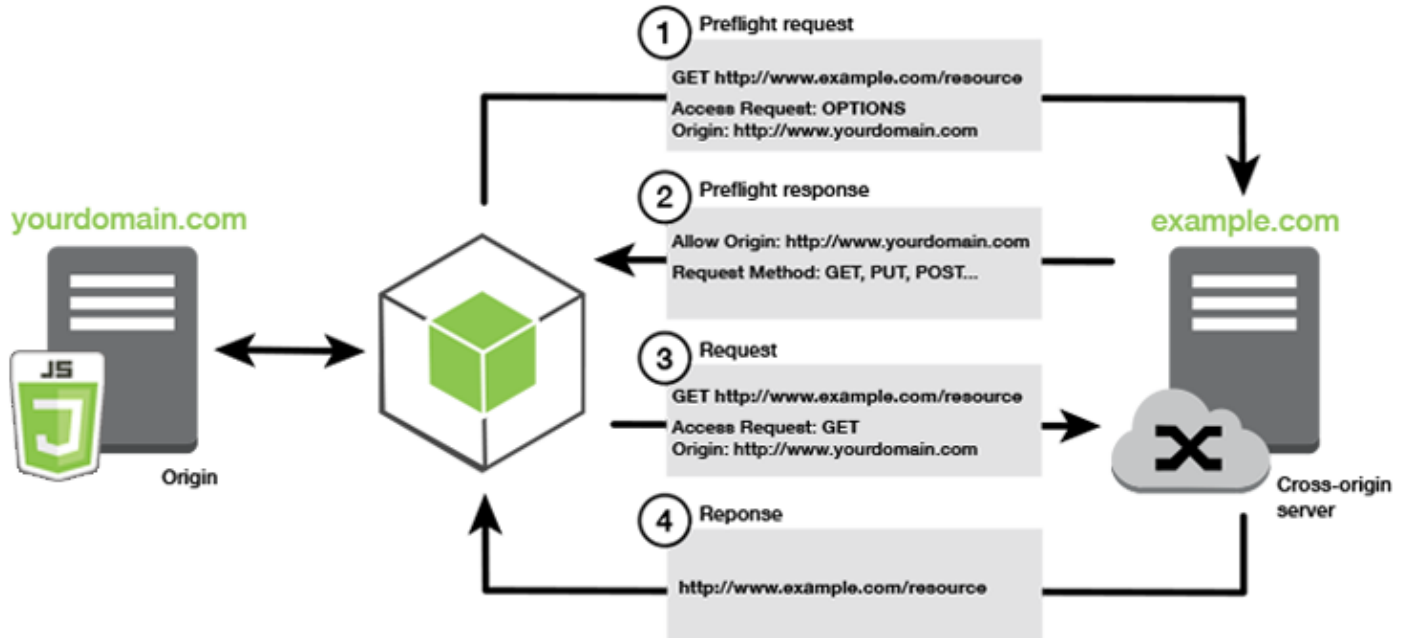
CORS는 다음을 기반으로 교차 오리진 요청 시 리소스 공유 허용 여부를 결정합니다.

- 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 CORS 는 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 필요합니까?

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 CORS 구성

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

AWS Web Services Management Console에서 CORS를 구성하는 경우 JSON을 사용하여 CORS 구성을 생성해야 합니다. 새로운 AWS Web Services Management Console은 JSON CORS 구성만 지원합니다.

⚠ Important

새 AWS Web Services Management Console에서 CORS 구성은 JSON이어야 합니다.

1. AWS Web Services Management Console에서 Amazon S3 콘솔을 열고, 구성하려는 버킷을 찾아 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 100 개까지 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.

Webpack으로 애플리케이션 번들링

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. SDK for JavaScript는 출력 번들에 포함할 종속성 중 하나로 webpack에 포함될 수 있습니다.

webpack에 관한 자세한 내용은 GitHub의 [webpack module bundler](#)를 참조하세요.

Webpack 설치

webpack 모듈 번들러를 설치하려면 먼저, Node.js 패키지 관리자인 npm이 설치되어 있어야 합니다. 다음 명령을 입력하여 webpack CLI 및 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
```

Webpack 구성

기본적으로 webpack은 프로젝트의 루트 디렉터리에서 `webpack.config.js`라는 JavaScript 파일을 검색합니다. 이 파일은 구성 옵션을 지정합니다. 다음은 WebPack 버전 5.0.0 이상에 대한 `webpack.config.js` 구성 파일의 예입니다.

Note

Webpack 구성 요구 사항은 설치하는 Webpack 버전에 따라 다릅니다. 자세한 내용은 [Webpack documentation](#)을 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 webpack이 가져온 모듈 검색을 시작하는 데 사용하는 파일입니다. 출력의 파일 이름은 `bundle.js`로 지정합니다. 출력 파일에는 애플리케이션에서 실행해야 하는 JavaScript가 모두 포함되어 있습니다. 진입점에 지정된 코드가 SDK for JavaScript와 같은 다른 모듈을 가져오거나 필요로 하는 경우 해당 코드는 구성에서 이를 지정할 필요 없이 번들링됩니다.

Webpack 실행

webpack을 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.

Webpack 번들 사용

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.js에 대한 번들링

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

SDK for JavaScript에서 AWS 서비스 작업

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 리소스를 사용하기 위한 하위 수준 APIs를 제공하는 클라이언트 클래스가 하나 이상 있습니다. 예를 들어 Amazon DynamoDB API는 DynamoDB 클래스를 통해 사용할 수 있습니다.

SDK for JavaScript를 통해 노출되는 서비스는 요청-응답 패턴에 따라 호출 애플리케이션과 메시지를 교환합니다. 이 패턴에서 서비스를 호출하는 코드는 해당 서비스의 엔드포인트에 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 재시도를 포함하여 서비스 객체에 대한 작업의 전체 요청 및 응답 수명 주기가 포함됩니다. 요청에는 0개 이상의 속성이 JSON 파라미터로 포함됩니다. 응답은 작업과 관련된 객체에 캡슐화되며 콜백 함수 또는 JavaScript promise와 같은 여러 기법 중 하나를 통해 요청자에게 반환됩니다.

주제

- [서비스 객체 생성 및 직접 호출](#)
- [비동기식 서비스 직접 호출](#)
- [서비스 클라이언트 요청 생성](#)
- [서비스 클라이언트 응답 처리](#)
- [JSON 작업](#)
- [AWS SDK for JavaScript 통화 로깅](#)
- [DynamoDB에서 AWS 계정 기반 엔드포인트 사용](#)
- [Amazon S3 체크섬을 통한 데이터 무결성 보호](#)
- [SDK for JavaScript 코드 예](#)

서비스 객체 생성 및 직접 호출

JavaScript API는 사용 가능한 대부분의 AWS 서비스를 지원합니다. JavaScript API의 각 서비스는 서비스에서 지원하는 모든 API를 간접적으로 호출하는 데 사용되는 send 메서드를 클라이언트 클

래스에 제공합니다. JavaScript API의 서비스 클래스, 작업 및 파라미터에 관한 자세한 내용은 [API Reference](#)를 확인하세요.

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 파라미터 지정

서비스 객체의 메서드를 호출할 때 API에서 요구하는 대로 파라미터를 JSON으로 전달합니다. 예를 들어 Amazon S3에서 지정된 버킷 및 키에 대한 객체를 가져오려면 `S3Client`의 `GetObjectCommand` 메서드에 다음 파라미터를 전달합니다. JSON 파라미터 전달에 대한 자세한 내용은 [JSON 작업](#) 섹션을 참조하세요.

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

Amazon S3 파라미터에 관한 자세한 내용은 API 참조의 [@aws-sdk/client-s3](#) 섹션을 참조하세요.

TypeScript에서 생성된 클라이언트에 @smithy/types 사용

TypeScript를 사용하는 경우 `@smithy/types` 패키지를 사용하여 클라이언트의 입력 및 출력 셰이프를 조작할 수 있습니다.

시나리오: 입력 및 출력 구조에서 **undefined** 제거

생성된 셰이프의 멤버는 입력 셰이프의 경우 `undefined`와 결합되고 출력 셰이프의 경우 `?`입니다(선택 사항). 입력의 경우 서비스에 대한 검증이 연기됩니다. 출력의 경우 출력 데이터를 런타임으로 확인하는 것이 좋습니다.

이 단계를 건너뛰려면 `AssertiveClient` 또는 `UncheckedClient` 유형 도우미를 사용합니다. 다음 예제에서는 Amazon S3 서비스와 함께 유형 도우미를 사용합니다.

```
import { S3 } from "@aws-sdk/client-s3";
import type { AssertiveClient, UncheckedClient } from "@smithy/types";

const s3a = new S3({}) as AssertiveClient<S3>;
const s3b = new S3({}) as UncheckedClient<S3>;

// AssertiveClient enforces required inputs are not undefined
// and required outputs are not undefined.
const get = await s3a.getObject({
  Bucket: "",
  // @ts-expect-error (undefined not assignable to string)
  Key: undefined,
});

// UncheckedClient makes output fields non-nullable.
// You should still perform type checks as you deem
// necessary, but the SDK will no longer prompt you
// with nullability errors.
const body = await (
  await s3b.getObject({
    Bucket: "",
    Key: "",
  })
).Body.transformToString();
```

Command 구문과 함께 집계되지 않은 클라이언트에서 트랜스포밍을 사용하는 경우 아래 예제와 같이 다른 클래스를 거치기 때문에 입력을 검증할 수 없습니다.

```
import { S3Client, ListBucketsCommand, GetObjectCommand, GetObjectCommandInput } from
"@aws-sdk/client-s3";
import type { AssertiveClient, UncheckedClient, NoUndefined } from "@smithy/types";

const s3 = new S3Client({}) as UncheckedClient<S3Client>;

const list = await s3.send(
  new ListBucketsCommand({
    // command inputs are not validated by the type transform.
    // because this is a separate class.
  })
);

/**
 * Although less ergonomic, you can use the NoUndefined<T>
```

```

    * transform on the input type.
    */
const getObjectInput: NoUndefined<GetObjectCommandInput> = {
    Bucket: "undefined",
    // @ts-expect-error (undefined not assignable to string)
    Key: undefined,
    // optional params can still be undefined.
    SSECustomerAlgorithm: undefined,
};

const get = s3.send(new GetObjectCommand(getObjectInput));

// outputs are still transformed.
await get.Body.TransformToString();

```

시나리오: Smithy-TypeScript 생성 클라이언트의 출력 페이로드 Blob 유형 좁히기

이 시나리오는 AWS SDK for JavaScript v3S3Client의 내에서와 같이 스트리밍 본문이 있는 작업과 대부분 관련이 있습니다.

Blob 페이로드 유형은 플랫폼에 따라 다르므로 애플리케이션에서 클라이언트가 특정 환경에서 실행 중임을 표시하도록 하고 싶을 수 있습니다. 이렇게 하면 다음 예제와 같이 Blob 페이로드 유형이 좁아 집니다.

```

import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import type { NodeJsClient, SdkStream, StreamingBlobPayloadOutputTypes } from "@smithy/
types";
import type { IncomingMessage } from "node:http";

// default client init.
const s3Default = new S3Client({});

// client init with type narrowing.
const s3NarrowType = new S3Client({}) as NodeJsClient<S3Client>;

// The default type of blob payloads is a wide union type including multiple possible
// request handlers.
const body1: StreamingBlobPayloadOutputTypes = (await s3Default.send(new
  GetObjectCommand({ Key: "", Bucket: "" })))
  .Body!;

// This is of the narrower type SdkStream<IncomingMessage> representing
// blob payload responses using specifically the node:http request handler.

```

```
const body2: SdkStream<IncomingMessage> = (await s3NarrowType.send(new
  GetObjectCommand({ Key: "", Bucket: "" })))
  .Body!;
```

비동기식 서비스 직접 호출

SDK를 통해 수행한 모든 요청은 비동기식입니다. 브라우저 스크립트를 작성할 때 이 점을 항상 주의해야 합니다. 웹 브라우저에서 실행 중인 JavaScript에는 일반적으로 실행 스레드가 하나 뿐입니다. AWS 서비스에 대한 비동기식 호출을 수행한 후 브라우저 스크립트는 계속 실행되며 프로세스에서 반환되기 전에 해당 비동기식 결과에 의존하는 코드를 실행하려고 시도할 수 있습니다.

AWS 서비스에 대한 비동기식 호출에는 해당 호출을 관리하여 데이터를 사용할 수 있기 전에 코드가 데이터를 사용하려고 하지 않도록 하는 것이 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/await`를 사용하는 것이 좋습니다.

`async/await`

이 기법은 V3의 기본 동작이므로 사용하는 것이 좋습니다.

`promise`

`async/await`를 지원하지 않는 브라우저에서 이 기법을 사용하세요.

`callback`

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

주제

- [비동기 직접 호출 관리](#)
- [비동기/대기 사용](#)
- [JavaScript Promise 사용](#)
- [익명 콜백 함수 사용](#)

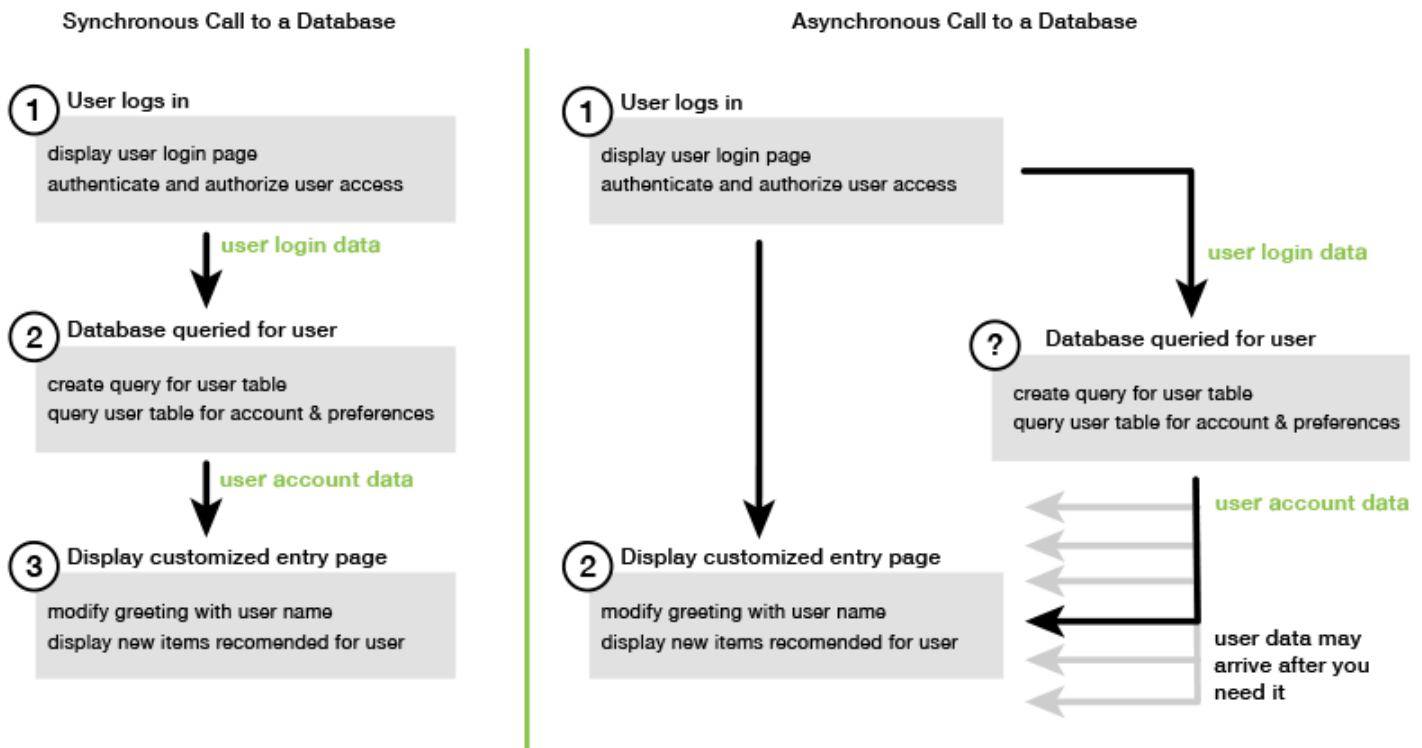
비동기 직접 호출 관리

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 보안 인증으로 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-2의 모든 Amazon DynamoDB 테이블을 나열합니다.

Note

이 예를 실행하려면 다음을 수행합니다.

- 프로젝트의 명령줄 `npm install @aws-sdk/client-dynamodb`을 입력하여 AWS SDK for JavaScript DynamoDB 클라이언트를 설치합니다.
- 자격 AWS 증명을 올바르게 구성했는지 확인합니다. 자세한 내용은 [자격 증명 설정](#) 단원을 참조하십시오.

```
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 functions](#)를 참조하세요.

JavaScript Promise 사용

서비스 클라이언트의 AWS SDK for JavaScript v3 메서드(ListTablesCommand)를 사용하여 콜백을 사용하는 대신 서비스를 호출하고 비동기 흐름을 관리합니다. 다음 예는 us-west-2에서 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 Promise 조정

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 개의 비동기 호출을 수행해야 하지만 각 호출에 대한 promise가 이행된 후에만 완료될 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

기본 JavaScript promise(ECMAScript 2015)에 대한 지원은 모드가 실행되는 JavaScript 엔진 및 버전에 따라 달라집니다. 코드를 실행해야 하는 각 환경에서 JavaScript promise에 대한 지원을 확인하려면 GitHub의 [ECMAScript compatibility table](#)을 참조하세요.

익명 콜백 함수 사용

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 for JavaScript 버전 3(V3)을 사용하면 요청을 보낼 수 있습니다.

Note

SDK for JavaScript V3를 사용하는 경우 버전 2(V2) 명령을 사용하여 작업을 수행할 수도 있습니다. 자세한 내용은 [v2 명령 사용](#) 단원을 참조하십시오.

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 *AbcInput*입니다.
3. 필요에 따라 요청 객체를 입력으로 사용하여 서비스 명령을 초기화합니다.
4. 명령 객체를 입력으로 사용하여 클라이언트에서 `send`를 직접적으로 호출합니다.

예를 들어 us-west-2의 Amazon DynamoDB 테이블을 나열하려면 `async/await`를 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 데이터 액세스

응답 객체에는 서비스 요청에서 반환한 데이터가 속성으로 포함됩니다.

[서비스 클라이언트 요청 생성](#)에서 `ListTablesCommand` 명령은 응답의 `TableNames` 속성에 테이블 이름을 반환했습니다.

오류 정보 액세스

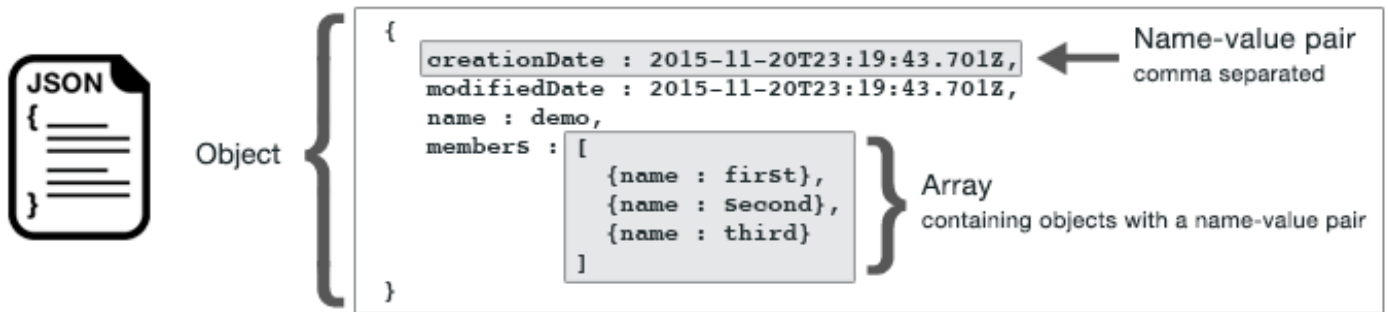
명령이 실패하면 예외가 발생합니다. 다음 코드 스니펫은 서비스 예외를 처리하는 방법을 보여줍니다.

```
try {  
  await client.send(someCommand);  
} catch (e) {  
  if (e.name === "InvalidSignatureException") {  
    // Handle InvalidSignatureException  
  } else if (e.name === "ResourceNotFoundException") {  
    // Handle ResourceNotFoundException  
  } else if (e.name === "FooServiceException") {  
    // Handle all other server-side exceptions from Foo service  
  } else {  
    // Handle errors from SDK  
  }  
}
```

JSON 작업

JSON은 인간과 머신이 둘 다 판독 가능한 데이터를 교환하기 위한 형식입니다. JSON이라는 이름이 JavaScript Object Notation의 약어이지만, JSON의 형식은 프로그래밍 언어와 관련이 없습니다.

는 요청을 할 때 JSON을 AWS SDK for JavaScript 사용하여 서비스 객체로 데이터를 전송하고 서비스 객체에서 JSON으로 데이터를 수신합니다. JSON에 대한 자세한 내용은 [json.org](https://www.json.org)를 참조하세요.



JSON은 다음 두 가지 방식으로 데이터를 나타냅니다.

- 객체: 순서가 지정되지 않은 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 };
};
```

AWS SDK for JavaScript 통화 로깅

AWS SDK for JavaScript 는 기본 제공 로거와 함께 계속되므로 SDK for JavaScript를 사용하여 수행한 API 호출을 로깅할 수 있습니다.

콘솔에서 로거를 켜고 로그 항목을 인쇄하려면 선택적 logger 파라미터를 사용하여 서비스 클라이언트를 구성합니다. 아래 예제에서는 추적 및 디버그 출력을 무시하면서 클라이언트 로깅을 활성화합니다.

```
new S3Client({
  logger: {
    ...console,
    debug(...args) {},
    trace(...args) {},
  },
});
```

```
});
```

미들웨어를 사용하여 요청 로깅

는 미들웨어 스택을 AWS SDK for JavaScript 사용하여 작업 호출의 수명 주기를 제어합니다. 스택의 각 미들웨어는 요청 객체를 변경한 후 다음 미들웨어를 직접적으로 호출합니다. 이렇게 하면 어떤 미들웨어가 직접적으로 호출되어 오류가 발생했는지 정확하게 확인할 수 있으므로 스택의 문제를 디버깅하는 것도 훨씬 쉬워집니다. 다음은 미들웨어를 사용한 요청 로깅의 예제입니다.

```
const client = new DynamoDB({ region: "us-west-2" });

client.middlewareStack.add(
  (next, context) => async (args) => {
    console.log("AWS SDK context", context.clientName, context.commandName);
    console.log("AWS SDK request input", args.input);
    const result = await next(args);
    console.log("AWS SDK request output:", result.output);
    return result;
  },
  {
    name: "MyMiddleware",
    step: "build",
    override: true,
  }
);

await client.listTables({});
```

위 예제에서는 미들웨어가 DynamoDB 클라이언트의 미들웨어 스택에 추가됩니다. 첫 번째 인수는 직접적으로 호출할 스택 내 다음 미들웨어인 `next`와 호출되는 작업에 대한 정보를 포함하는 객체 `context`를 수락하는 함수입니다. 작업 및 요청에 전달된 파라미터가 포함된 객체인 `args`를 수락하는 함수를 반환하고 `args`를 사용하여 다음 미들웨어를 직접적으로 호출한 결과를 반환합니다.

DynamoDB에서 AWS 계정 기반 엔드포인트 사용

DynamoDB는 [AWS 계정 ID를 사용하여 요청 라우팅을 간소화하여 성능을 개선할 수 있는 계정 기반 엔드포인트](#)를 제공합니다. AWS

이 기능을 활용하려면 버전 3.656.0 이상의 AWS SDK for JavaScript 버전 3을 사용해야 합니다. 이 계정 기반 엔드포인트 기능은 이 새 버전에서 기본적으로 활성화됩니다.

계정 기반 라우팅을 옵트아웃하려는 경우 다음과 같은 옵션이 있습니다.

- `accountIdEndpointMode` 파라미터가 `disabled`로 설정된 DynamoDB 서비스 클라이언트를 구성합니다.
- 환경 변수 `AWS_ACCOUNT_ID_ENDPOINT_MODE`를 `disabled`로 설정합니다.
- 공유 AWS 구성 파일 설정을 `account_id_endpoint_mode`로 업데이트합니다 `disabled`.

다음 코드 스니펫은 DynamoDB 서비스 클라이언트를 구성하여 계정 기반 라우팅을 비활성화하는 방법의 예입니다.

```
const ddbClient = new DynamoDBClient({
  region: "us-west-2",
  accountIdEndpointMode: "disabled" // Disable account ID in the endpoint
});
```

AWS SDKs 및 도구 참조 가이드는 [다른 구성 옵션](#)에 대한 자세한 정보를 제공합니다.

Amazon S3 체크섬을 통한 데이터 무결성 보호

Amazon Simple Storage Service(S3)는 객체를 업로드할 때 체크섬을 지정하는 기능을 제공합니다. 체크섬을 지정하면 객체와 함께 저장되며 객체를 다운로드할 때 유효성을 검사할 수 있습니다.

체크섬은 파일을 전송할 때 데이터 무결성을 한층 더 강화합니다. 체크섬을 사용하면 수신된 파일이 원본 파일과 일치하는지 확인하여 데이터 일관성을 확인할 수 있습니다. Amazon S3의 체크섬에 대한 자세한 내용은 [지원되는 알고리즘](#)을 포함한 [Amazon Simple Storage Service 사용 설명서](#)를 참조하세요.

필요에 가장 적합한 알고리즘을 유연하게 선택하고 SDK가 체크섬을 계산하도록 할 수 있습니다. 또는 지원되는 알고리즘 중 하나를 사용하여 미리 계산된 체크섬 값을 제공할 수 있습니다.

Note

AWS SDK for JavaScript의 버전 3.729.0부터 SDK는 업로드에 대한 CRC32 체크섬을 자동으로 계산하여 기본 무결성 보호를 제공합니다. 사전 계산된 체크섬 값을 제공하지 않거나 SDK가 체크섬을 계산하는 데 사용해야 하는 알고리즘을 지정하지 않은 경우 SDK는 이 체크섬을 계산합니다.

또한 SDK는 [AWS SDK 및 도구 참조 안내서](#)에서 읽을 수 있는 외부에서 설정할 수 있는 데이터 무결성 보호에 대한 전역 설정을 제공합니다.

객체 업로드

S3Client의 [PutObject](#) 명령을 사용하여 Amazon S3에 객체를 업로드합니다. 빌더의 `ChecksumAlgorithm` 파라미터를 사용하여 `PutObjectRequest`에서 체크섬 계산을 활성화하고 알고리즘을 지정합니다. 유효한 값은 [지원되는 체크섬 알고리즘](#)을 참조하세요.

다음 코드 스니펫은 CRC-32 체크섬이 있는 객체를 업로드하라는 요청을 보여줍니다. SDK는 요청을 보내면 CRC-32 체크섬을 계산하고 객체를 업로드합니다. Amazon S3은 객체와 함께 체크섬을 저장합니다.

```
import { ChecksumAlgorithm, S3 } from "@aws-sdk/client-s3";

const client = new S3();
const response = await client.putObject({
  Bucket: "my-bucket",
  Key: "my-key",
  Body: "Hello, world!",
  ChecksumAlgorithm: ChecksumAlgorithm.CRC32,
});
```

요청에 체크섬 알고리즘을 제공하지 않는 경우 체크섬 동작은 다음 표와 같이 사용하는 SDK 버전에 따라 달라집니다.

체크섬 알고리즘이 제공되지 않은 경우 체크섬 동작

SDK for JavaScript 버전	체크섬 동작
3.729.0 이전	SDK는 CRC 기반 체크섬을 자동으로 계산하여 요청에 제공하지 않습니다.
3.729.0 이상	SDK는 CRC32 알고리즘을 사용하여 체크섬을 계산하고 요청에 제공합니다. Amazon S3는 자체 CRC32 체크섬을 계산하여 전송의 무결성을 확인하고 이를 SDK에서 제공하는 체크섬과 비교합니다. 체크섬이 일치하면 체크섬이 객체와 함께 저장됩니다.

SDK에서 계산하는 체크섬이 Amazon S3가 요청을 수신할 때 계산하는 체크섬과 일치하지 않는 경우 오류가 반환됩니다.

미리 계산된 체크섬 값 사용

요청과 함께 제공되는 사전 계산된 체크섬 값은 SDK의 자동 계산을 비활성화하고 제공된 값을 대신 사용합니다.

다음 예제는 사전 계산된 SHA-256 체크섬을 포함하는 요청을 보여줍니다.

```
import { S3 } from "@aws-sdk/client-s3";
import { createHash } from "node:crypto";

const client = new S3();

const Body = "Hello, world!";
const ChecksumSHA256 = await createHash("sha256").update(Body).digest("base64");

const response = await client.putObject({
  Bucket: "my-bucket",
  Key: "my-key",
  Body,
  ChecksumSHA256,
});
```

Amazon S3에서 체크섬 값이 지정된 알고리즘에 대해 올바르지 않다고 판단하면 서비스는 오류 응답을 반환합니다.

멀티파트 업로드

멀티파트 업로드에 체크섬을 사용할 수도 있습니다. `Upload` 라이브러리 옵션을 사용하여 멀티파트 업로드@aws-sdk/lib-storage와 함께 체크섬을 사용할 AWS SDK for JavaScript 수 있습니다.

```
import { ChecksumAlgorithm, S3 } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";
import { createReadStream } from "node:fs";

const client = new S3();
const filePath = "/path/to/file";
const Body = createReadStream(filePath);

const upload = new Upload({
  client,
  params: {
    Bucket: "my-bucket",
```

```
    Key: "my-key",
    Body,
    ChecksumAlgorithm: ChecksumAlgorithm.CRC32,
  },
});
await upload.done();
```

SDK for JavaScript 코드 예

이 섹션의 주제에는 다양한 서비스의 API와 AWS SDK for JavaScript 함께를 사용하여 일반적인 작업을 수행하는 방법에 대한 예제가 포함되어 있습니다. APIs

[GitHub의AWS 코드 예 리포지토리](#)에서 이러한 예의 소스 코드와 다른 소스 코드를 찾아보세요. AWS 설명서 팀이 생성을 고려할 새 코드 예제를 제안하려면 요청을 생성합니다. 이 팀은 개별 API 호출만 다루는 간단한 코드 조각에 비해 광범위한 시나리오 및 사용 사례를 다루는 코드를 생성하려고 합니다. 지침은 [GitHub Contributing Guidelines](#)의 코드 작성 섹션을 참조하세요.

Important

이러한 예에서는 ECMAScript6 import/export 구문을 사용합니다.

- 따라서 Node.js 버전 14.17 이상이 필요합니다. 최신 버전의 Node.js를 다운로드하여 설치하려면 [Node.js downloads](#)를 참조하세요.
- CommonJS 구문을 사용하려는 경우 변환 지침은 [JavaScript ES6/CommonJS 구문](#) 단원을 참조하세요.

주제

- [JavaScript ES6/CommonJS 구문](#)
- [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](#)
- [API Gateway를 사용하여 Lambda 호출](#)
- [AWS Lambda 함수를 실행하기 위한 예약된 이벤트 생성](#)
- [Amazon Lex 챗봇 빌드](#)

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 downloads](#)를 참조하세요. 그러나 원하는 경우 다음 지침을 사용하여 코드 예를 CommonJS 구문으로 변환할 수 있습니다.

- 프로젝트 환경의 package.json에서 "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();
```

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 설명은 [AWS Elemental MediaConvert사용자 가이드](#)를 참조하세요.

MediaConvert용 JavaScript API는 MediaConvert 클라이언트 클래스를 통해 노출됩니다. 자세한 내용은 API 참조의 [Class: MediaConvert](#)를 참조하세요.

주제

- [MediaConvert에서 트랜스코딩 작업 생성 및 관리](#)
- [MediaConvert에서 작업 템플릿 사용](#)

MediaConvert에서 트랜스코딩 작업 생성 및 관리



이 Node.js 코드 예제는 다음을 보여 줍니다.

- 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. [GitHub](#)의 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 downloads](#)를 참조하세요. 그러나 CommonJS 구문을 사용하려는 경우 [JavaScript ES6/CommonJS 구문](#) 단원을 참조하세요.

단순 트랜스코딩 작업 정의

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 콘솔](#)을 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 바꾸고, *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 data에서 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

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_ID를 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

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_ARN을 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. [GitHub](#)의 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 downloads](#)를 참조하세요.

그러나 CommonJS 구문을 사용하려는 경우 [JavaScript ES6/CommonJS 구문](#) 단원을 참조하세요.

작업 템플릿 생성

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 관리](#)의 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

파일 이름이 `emc_template_createjob.js`인 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 바꾸고, *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

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

`listTemplates` 클라이언트 클래스의 `MediaConvert` 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 `promise`를 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

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

삭제하려는 작업 템플릿의 이름을 `DeleteJobTemplateCommand` 클라이언트 클래스의 `MediaConvert` 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 실행하기 위한 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.

- [GitHub](#)의 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) 인증되지 않은 Amazon Cognito 사용자 역할 polly:SynthesizeSpeech 권한과 IAM 역할이 연결된 Amazon Cognito 자격 증명 풀을 생성합니다. 아래 [를 사용하여 AWS 리소스 생성 CloudFormation](#) 단원에서는 이러한 리소스를 생성하는 방법을 설명합니다.

Note

이 예제에서는 Amazon Cognito를 사용하지만 Amazon Cognito를 사용하지 않는 경우 AWS 사용자에게 다음 IAM 권한 정책이 있어야 합니다.

JSON

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

를 사용하여 AWS 리소스 생성 CloudFormation

CloudFormation 를 사용하면 AWS 인프라 배포를 예측 가능하고 반복적으로 생성하고 프로비저닝할 수 있습니다. 에 대한 자세한 내용은 [AWS CloudFormation 사용 설명서를](#) CloudFormation참조하세요.

CloudFormation 스택을 생성하려면:

1. [AWS CLI 사용 설명서](#)의 AWS CLI 지침에 따라를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 디렉터리에 이름이 setup.yaml인 파일을 생성하고 [여기 GitHub의](#) 내용을 해당 파일에 복사합니다.

Note

CloudFormation 템플릿은 [GitHub에서 사용할](#) 수 있는 AWS CDK 사용하여 생성되었습니다. 에 대한 자세한 내용은 [AWS Cloud Development Kit \(AWS CDK\) 개발자 안내서](#)를 AWS CDK참조하세요.

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

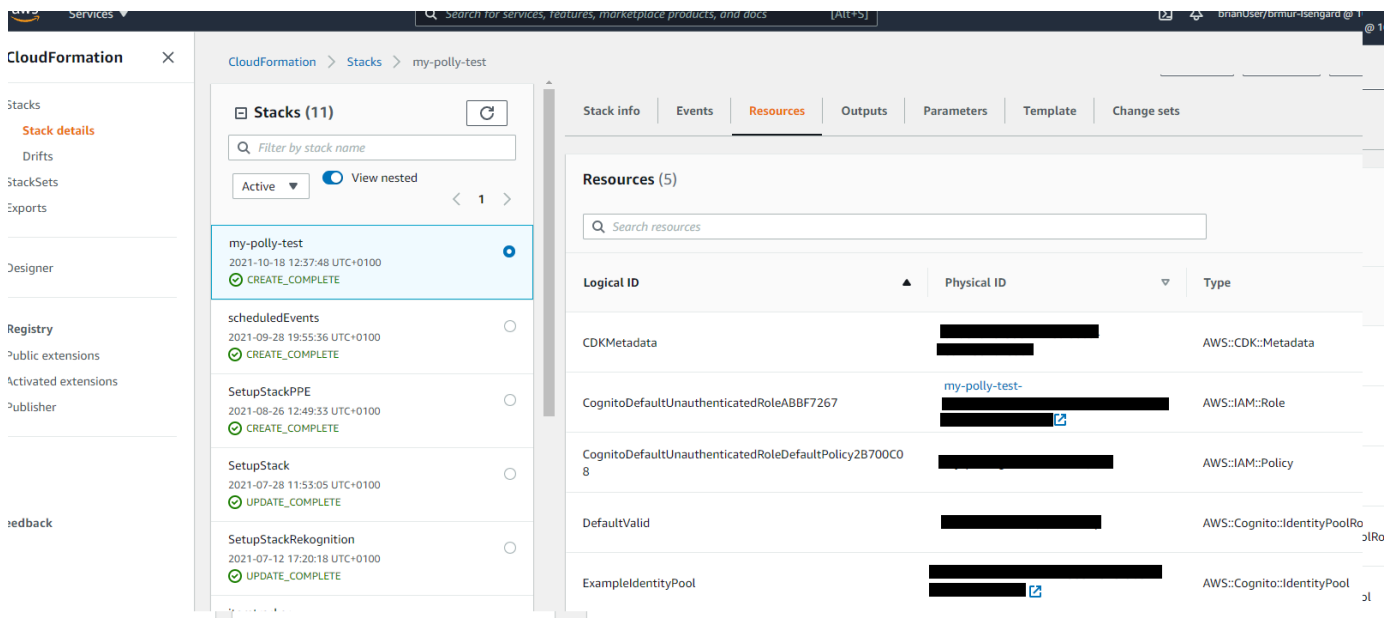
Important

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 [CloudFormation 사용 설명서](#)를 참조하세요.

- CloudFormation 관리 콘솔로 이동하여 스택을 선택하고 스택 이름을 선택한 다음 리소스 탭을 선택하여 생성된 리소스 목록을 봅니다.



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.

```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "../libs/pollyClient.js";

// Create the parameters
const 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);
  }
};

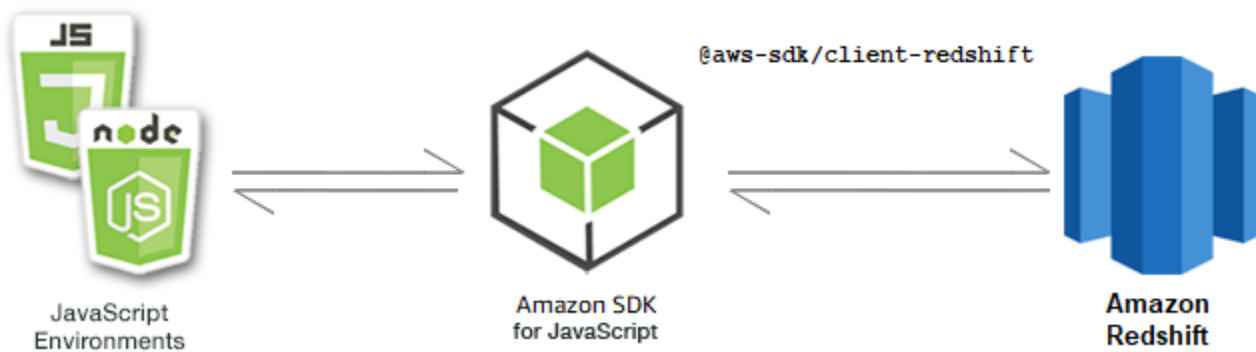
run();
```

이 샘플 코드는 [GitHub](#)에서 찾을 수 있습니다.

Amazon Redshift 예

Amazon Redshift는 클라우드에서 완전히 관리되는 페타바이트급 데이터 웨어하우스 서비스입니다. Amazon Redshift 데이터 웨어하우스는 노드라는 컴퓨팅 리소스의 모음으로, 노드는 클러스터라는 그

를 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 안내서](#)를 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. [GitHub](#)의 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 downloads](#)를 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.js인 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.js인 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_NAME을 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-node를 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 [GitHub](#)에서 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.js인 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.js인 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_NAME을 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 [GitHub](#)에서 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.js인 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.js인 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_NAME을 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 [GitHub](#)에서 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.js인 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_NAME을 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();
```

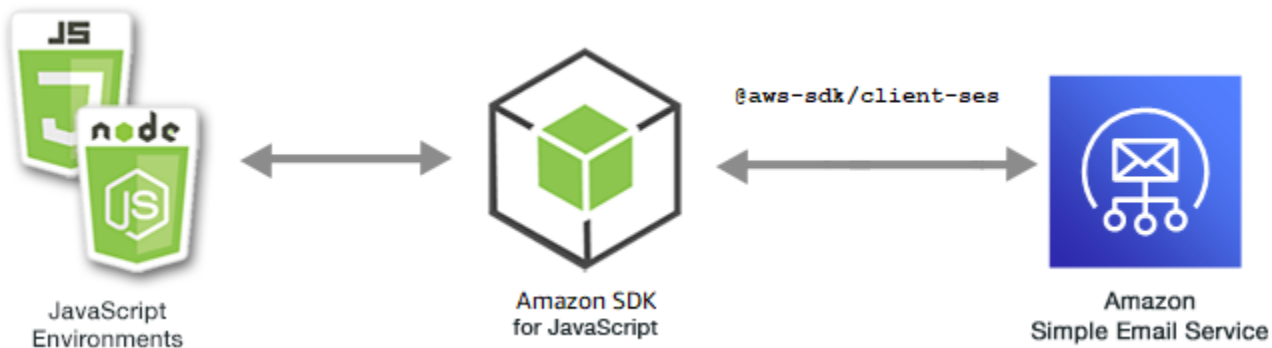
예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 샘플 코드는 [GitHub](#)에서 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 SES용 JavaScript API는 SES 클라이언트 클래스를 통해 노출됩니다. Amazon SES 클라이언트 클래스 사용에 관한 자세한 내용은 API 참조의 [Class: SES](#)를 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
- AWS Identity and Access Management(IAM) 정책을 Amazon SES 자격 증명에 할당하는 방법
- AWS 계정의 모든 Amazon SES 자격 증명을 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. 이 Node.js 모듈은 SDK for JavaScript에서 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 노드 TypeScript 예를 실행하도록 프로젝트 환경을 설정하고 필수 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 downloads](#)를 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. **REGION**을 AWS 리전으로 바꿉니다.

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

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

반환된 data에는 IdentityType 파라미터로 지정된 도메인 자격 증명 배열이 포함되어 있습니다.

📌 Note

IdentityType을 자격 증명 유형("EmailAddress" 또는 "Domain"일 수 있음)으로 바꿉니다.

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

이메일 주소 자격 증명 확인

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

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

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

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

 Note

EMAIL_ADDRESS를 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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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. **REGION**을 AWS 리전으로 바꿉니다.

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

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

Note

이 예에서는 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](#)을 참조하세요.

Note

DOMAIN_NAME을 도메인 이름으로 바꿉니다.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utlis/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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

자격 증명 삭제

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

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

```

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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

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

Note

이 예에서는 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](#)을 참조하세요.

Note

IDENTITY_EMAIL을 삭제할 자격 증명의 이메일로 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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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. 이 Node.js 모듈은 SDK for JavaScript에서 SES 클라이언트 클래스의 다음 메서드를 사용하여 이메일 템플릿을 생성하고 사용합니다.

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

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. [GitHub](#)의 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 downloads](#)를 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SES 클라이언트 객체가 생성됩니다. **REGION**을 AWS 리전으로 바꿉니다.

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

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

Note

이 예에서는 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](#)을 참조하세요.

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

이메일 템플릿 가져오기

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

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

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

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

Note

이 예에서는 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](#)을 참조하세요.

Note

TEMPLATE_NAME을 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 (caught) {
  if (caught instanceof Error && caught.name === "MessageRejected") {
    /** @type { import('@aws-sdk/client-ses').MessageRejected } */
    const messageRejectedError = caught;
    return messageRejectedError;
  }
  throw caught;
}
};

```

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

```
node ses_gettemplate.js
```

이 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

이메일 템플릿 생성

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

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

```

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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

CreateTemplateCommand, SES, TemplateName 및 HtmlPart를 포함하여 SubjectPart 클라이언트 클래스의 TextPart 메서드에 대한 파라미터를 전달할 객체를 생성합니다.

CreateTemplateCommand 메서드를 직접적으로 호출하려면 Amazon SES 클라이언트 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

Note

이 예에서는 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](#)을 참조하세요.

Note

TEMPLATE_NAME을 새 템플릿의 이름으로 바꾸고, **HtmlPart**를 이메일의 HTML 태그 지정 콘텐츠를 바꾸고, **SubjectPart**를 이메일 제목으로 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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

이메일 템플릿 업데이트

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

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

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

Template 클라이언트 클래스의 TemplateName 메서드에 전달된 필수 UpdateTemplateCommand 파라미터와 함께 템플릿에서 업데이트하려는 SES 파라미터 값을 전달할 객체를 생성합니다.

UpdateTemplateCommand 메서드를 직접적으로 호출하려면 Amazon SES 서비스 객체를 간접적으로 호출하여 파라미터를 전달합니다.

Note

이 예에서는 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](#)을 참조하세요.

Note

TEMPLATE_NAME을 템플릿 이름으로 바꾸고 **HTML_PART**를 HTML 태그가 지정된 이메일 콘텐츠로 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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

이메일 템플릿 삭제

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

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

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

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

Note

이 예에서는 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](#)을 참조하세요.

Note

TEMPLATE_NAME을 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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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. 이 Node.js 모듈은 SDK for JavaScript에서 SES 클라이언트 클래스의 다음 메서드를 사용하여 이메일 템플릿을 생성하고 사용합니다.

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

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. [GitHub](#)의 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 downloads](#)를 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. **REGION**을 AWS 리전으로 바꿉니다.

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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 예에서는 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](#)을 참조하세요.

Note

*toAddress*를 이메일 수신자의 주소로 바꾸고, *fromAddress*를 이메일 발신자의 이메일 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 (caught) {
        if (caught instanceof Error && caught.name === "MessageRejected") {
            /** @type { import('@aws-sdk/client-ses').MessageRejected } */
            const messageRejectedError = caught;
            return messageRejectedError;
        }
        throw caught;
    }
};

```

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

```
node ses_sendemail.js
```


이 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

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

Note

이 예에서는 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](#)을 참조하세요.

Note

REGION을 AWS 리전으로 바꾸고, **USER**를 이메일 수신자의 이름과 이메일 주소로, **VERIFIED_EMAIL**를 이메일 발신자의 이메일 주소로, **TEMPLATE_NAME**을 템플릿 이름으로 바꿉니다.

```
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 (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};

```

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

```
node ses_sendtemplatedemail.js
```

이 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

템플릿을 사용한 대량 이메일 전송

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

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

```
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 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

파일 이름이 ses_sendbulktemplatedemail.js인 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 예에서는 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](#)을 참조하세요.

Note

USERS를 이메일 수신자의 이름과 이메일 주소로 바꾸고, **VERIFIED_EMAIL_1**을 이메일 발신자의 이메일 주소로 바꾸고, **TEMPLATE_NAME**을 템플릿 이름으로 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 (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};

```

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

```
node ses_sendbulktemplatedemail.js
```

이 코드 예는 [여기 GitHub에서](#) 찾을 수 있습니다.

Amazon Simple Notification Service 예

Amazon Simple Notification Service(Amazon SNS)는 구독 엔드포인트 또는 클라이언트에 대한 메시지 전송 또는 전송을 조정하고 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: SNS](#)를 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 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 downloads](#)를 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

SNS 클라이언트 클래스의 CreateTopicCommand 메서드에 새 주제의 Name을 전달할 객체를 생성합니다. CreateTopicCommand 메서드를 직접적으로 호출하려면 Amazon SNS 서비스 객체를 간접적으로 호출하는 비동기 함수를 생성하여 파라미터 객체를 전달합니다. 반환되는 data에는 주제의 ARN이 포함됩니다.

Note

TOPIC_NAME을 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 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 [GitHub](#)에서 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 해당 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.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성합니다.

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

Note

TOPIC_ARN을 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

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

Note

TOPIC_ARN을 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: 'xxxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};

```

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 해당 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_NAME을 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 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 downloads](#)를 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 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 [SetSMSAttributes](#)를 참조하세요.

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

Note

MESSAGE_TEXT를 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-xxxxxxxxxxxxxx'  
// }  
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 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 downloads](#)를 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 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_ARN을 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: 'corepyale@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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

email 프로토콜, 구독할 주제의 TopicArn, 메시지 Endpoint로 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_ARN을 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 = "user@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

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 해당 AWS 리전으로 바꿉니다.

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

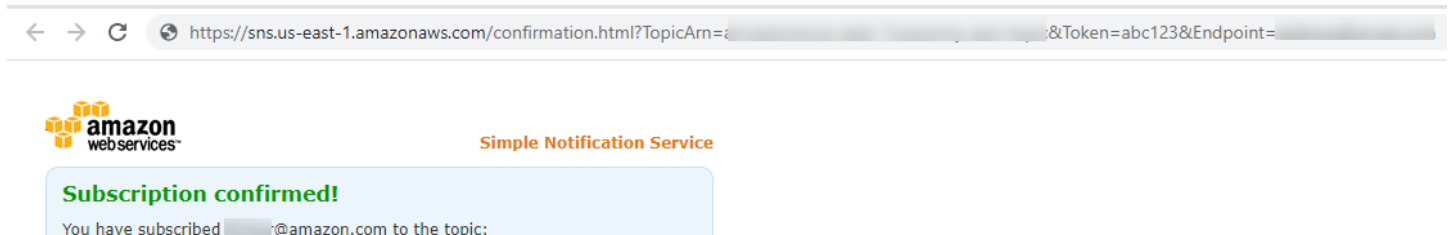
// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

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_ARN을 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-
  xxx-xxxx-xxxxxxxxxxxxx'
  // }
  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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 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_ARN을 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 예제에서는 Node.js 모듈을 사용하여 Amazon SNS 주제에서 알림을 수신하도록 AWS Lambda 함수를 구독합니다.

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 해당 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.js인 Node.js 모듈을 생성합니다. 위와 같이 SDK를 구성합니다.

Protocol 파라미터가 포함된 객체를 생성하여 lambda 프로토콜, 구독할 주제TopicArn의, 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_ARN을 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js`인 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 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 downloads](#)를 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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 [SetSMSAttributes](#)를 참조하세요.

이 예제에서는 SMS 메시지를 최저 비용이 발생하도록 메시지 전송을 최적화하는 Promotional로 전송할지 또는 최고 안정성을 달성하도록 메시지 전송을 최적화하는 Transactional로 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 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 [SetSMSAttributes](#)를 참조하세요.

다음 예제에서는 DefaultSMSType 속성을 Transactional로 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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_NUMBER를 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon SNS 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js`인 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_MESSAGE를 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-xxxxxxxxxxxxxx'
// }
return response;
};

```

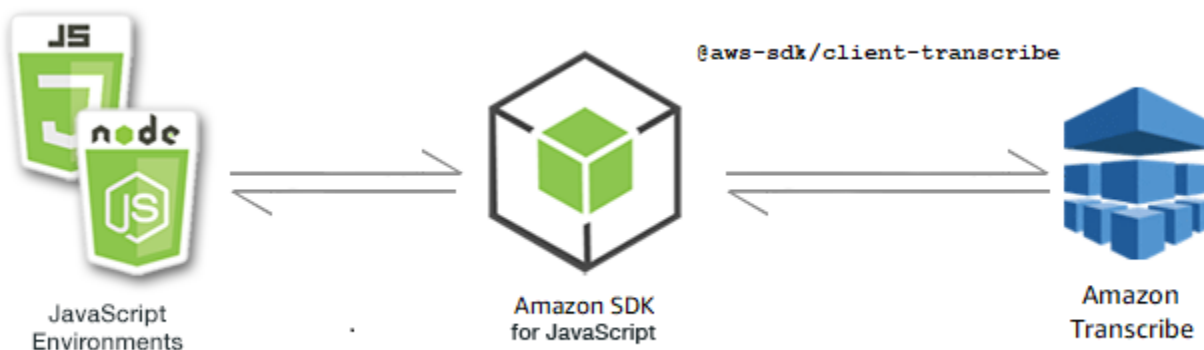
예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

```
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 사용자에게 관한 자세한 내용은 [Amazon Transcribe 개발자 안내서](#)를 참조하세요.

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 downloads](#)를 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 [StartTranscriptionJobCommand](#)를 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 해당 AWS 리전으로 바꿉니다.

```
import { TranscribeClient } from "@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.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 작업을 시작합니다.

Note

MEDICAL_JOB_NAME을 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 [GitHub](#)에서 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. **REGION**을 해당 AWS 리전으로 바꿉니다.

```
import { TranscribeClient } from "@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_WORD를 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 [GitHub](#)에서 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 Node.js 모듈을 생성합니다. 필수 클라이언트 및 패키지 설치를 포함하여 앞서 나와 있는 것처럼 SDK를 구성해야 합니다. AWS 리전과 삭제할 작업의 이름을 지정합니다.

Note

JOB_NAME을 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, 'transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
```

```
    new DeleteTranscriptionJobCommand(params),
  );
  console.log("Success - deleted");
  return data; // For unit tests.
} catch (err) {
  console.log("Error", err);
}
};
run();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 샘플 코드는 [GitHub](#)에서 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 사용자에게 관한 자세한 내용은 [Amazon Transcribe 개발자 안내서](#)를 참조하세요.

사전 필수 작업

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 downloads](#)를 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.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 의료 작업을 시작합니다.

i Note

MEDICAL_JOB_NAME을 의료 트랜스크립션 작업 이름으로 바꿉니다.

OUTPUT_BUCKET_NAME에 출력이 저장되는 Amazon S3 버킷을 지정합니다. **JOB_TYPE**에 작업 유형을 지정합니다. **SOURCE_LOCATION**에 소스 파일의 위치를 지정합니다.

SOURCE_FILE_LOCATION에 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 [GitHub](#)에서 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. **REGION**을 해당 AWS 리전으로 바꿉니다.

```
import { TranscribeClient } from "@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.js인 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

KEYWORD를 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 [GitHub](#)에서 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.js인 Node.js 모듈을 생성합니다. 이 모듈에 아래 코드를 복사하여 붙여 넣으면 Amazon Transcribe 클라이언트 객체가 생성됩니다. **REGION**을 해당 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.js인 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_NAME을 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_transcription_demo'
};

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

run();
```

예를 실행하려면 명령 프롬프트에서 다음을 입력합니다.

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

이 샘플 코드는 [GitHub](#)에서 찾을 수 있습니다.

Amazon EC2 인스턴스에서 Node.js 설정

SDK for JavaScript와 함께 Node.js를 사용하는 일반적인 시나리오는 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스에서 Node.js 웹 애플리케이션을 설정하고 실행하는 것입니다. 이 자습

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

사전 조건

이 자습서에서는 인터넷에서 접근 가능하고 SSH를 사용하여 연결할 수 있는 퍼블릭 DNS 이름으로 Linux 인스턴스를 이미 시작했다고 가정합니다. 자세한 내용은 Amazon EC2 사용 설명서의 [1단계: 인스턴스 시작](#)을 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 사용 설명서의 [Amazon EC2 사용 설정](#) 섹션을 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 는 다음 코드를 제어하지 않습니다. 실행하기 전에 먼저 신뢰성과 무결성을 확인해야 합니다. 이 코드에 대한 자세한 내용은 [nvm](#) GitHub 리포지토리에서 확인할 수 있습니다.

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

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

3. 명령줄에 다음을 입력하여 nvm을 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 설치의 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 사용 설명서의 [Amazon EBS 지원 Linux AMI 생성](#)을 참조하세요.

관련 리소스

이 주제에 사용되는 명령과 소프트웨어에 관한 자세한 내용은 다음 웹 페이지를 확인하세요.

- 노드 버전 관리자(nvm) - [GitHub의 nvm 리포지토리](#)를 참조하세요.
- 노드 패키지 관리자(npm) - [npm 웹 사이트](#)를 참조하세요.

API Gateway를 사용하여 Lambda 호출

대규모로 REST, HTTP 및 WebSocket API를 생성, 게시, 유지 관리, 모니터링 및 보호하기 위한 서비스인 Amazon API Gateway를 사용하여 Lambda 함수를 호출할 수 있습니다. AWS APIs API 개발자는 AWS 또는 기타 웹 서비스에 액세스하는 APIs와 AWS 클라우드에 저장된 데이터를 생성할 수 있습니다. API Gateway 개발자는 자체 클라이언트 애플리케이션에서 사용할 API를 생성할 수 있습니다. 자세한 내용은 [Amazon API Gateway란 무엇입니까?](#) 단원을 참조하세요.

AWS Lambda 는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있는 컴퓨팅 서비스입니다. 다양한 프로그래밍 언어로 Lambda 함수를 생성할 수 있습니다. 에 대한 자세한 내용은 [란 AWS Lambda](#) 무엇입니까?를 AWS Lambda참조하십시오.

이 예제에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 다양한 AWS 서비스를 호출하여 특정 사용 사례를 수행합니다. 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 사용하여이 AWS Lambda 함수를 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 이러한 필드가 포함된 Employee이라는 Amazon DynamoDB 테이블을 사용합니다.

- id - 테이블의 프라이머리 키입니다.
- firstName - 직원의 이름입니다.
- phone - 직원의 전화번호입니다.
- startDate - 직원의 시작 날짜입니다.

Scan [Table] Employee: Id + Add filter Start search					
☐	Id ⓘ	first	phone	startDate	
☐	1	Scott	15555555654	2019-12-20	
☐	2	Malcolm	15555555654	2019-12-17	
☐	55	Lam	15555555654	2019-12-19	

⚠ 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 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 Id라는 키가 있는 Employee라는 Amazon DynamoDB 테이블. 이 사용 사례를 테스트하려는 유효한 휴대폰을 포함해 올바른 데이터를 입력했는지 확인하세요. 자세한 내용은 [테이블 생성](#)을 참조하세요.
- Lambda 함수를 실행하기 위한 권한이 연결된 IAM 역할.
- Lambda 함수를 호스팅하는 Amazon S3 버킷.

이러한 리소스를 수동으로 생성할 수 있지만 이 자습서에 설명된 CloudFormation 대로를 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

를 사용하여 AWS 리소스 생성 CloudFormation

CloudFormation 를 사용하면 AWS 인프라 배포를 예측 가능하고 반복적으로 생성하고 프로비저닝할 수 있습니다. 에 대한 자세한 내용은 [AWS CloudFormation 사용 설명서를](#) CloudFormation참조하세요.

를 사용하여 CloudFormation 스택을 생성하려면 AWS CLI:

- [AWS CLI 사용 설명서](#)의 AWS CLI 지침에 따라를 설치하고 구성합니다.
- 프로젝트 폴더의 루트 디렉터리에 이름이 setup.yaml인 파일을 생성하고 [여기 GitHub의](#) 내용을 해당 파일에 복사합니다.

Note

CloudFormation 템플릿은 [GitHub에서 사용할](#) 수 있는 AWS CDK 사용하여 생성되었습니다. 에 대한 자세한 내용은 [AWS Cloud Development Kit \(AWS CDK\) 개발자 안내서](#)를 AWS CDK참조하세요.

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 [CloudFormation 사용 설명서](#)를 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: "1555555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "1555555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "1555555555555652" },
            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.js인 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 };

```

REGION을 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 };
```

REGION을 AWS 리전으로 바꿉니다. 이 코드는 [여기 GitHub에서](#) 제공됩니다.

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.)

```
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 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.)

```
// 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));
    for (const element of data.Items) {
      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 주제에서는 mylambdafunction.ts 및 이 예제에 필요한 AWS SDK for JavaScript 모듈을 라는 번들 파일로 번들링하는 방법을 설명합니다 index.js.

1. webpack을 아직 설치하지 않았다면 이 예의 [사전 필수 작업](#)에 따라 설치합니다.

Note

Webpack에 관한 자세한 내용은 [Webpack으로 애플리케이션 번들링](#) 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.js`인 것에 주목하세요. 이는 Lambda 함수가 작동하려면 `index.js` 핸들러가 있어야 하기 때문입니다.

3. 번들 출력 파일, `index.js`를 `mylambdafunction.zip`이라는 ZIP 파일로 압축합니다.
4. 이 자습서의 [AWS 리소스 생성](#) 항목에서 생성한 Amazon S3 버킷에 `mylambdafunction.zip`을 업로드합니다.

Lambda 함수 배포

프로젝트의 루트에서 `lambda-function-setup.ts` 파일을 생성하고 아래 내용을 해당 파일에 붙여 넣습니다.

BUCKET_NAME을 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

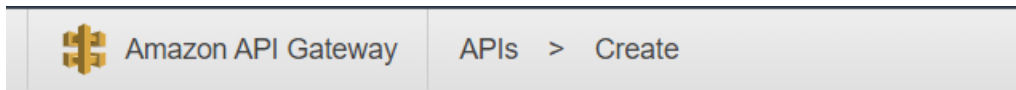
앱을 빌드하려면 다음을 수행합니다.

1. [REST API 생성](#)
2. [API Gateway 메서드 테스트](#)
3. [API Gateway 메서드 배포](#)

REST API 생성

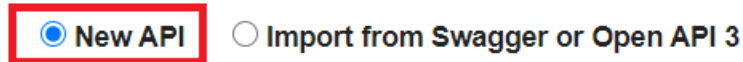
API Gateway 콘솔을 사용하여 Lambda 함수의 REST 엔드포인트를 생성할 수 있습니다. 완료하면 RESTful 호출을 사용하여 Lambda 함수를 간접적으로 호출할 수 있습니다.

1. [Amazon API Gateway 콘솔](#)에 로그인합니다.
2. REST API에서 빌드를 선택합니다.
3. 새 API를 선택합니다.



Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods.



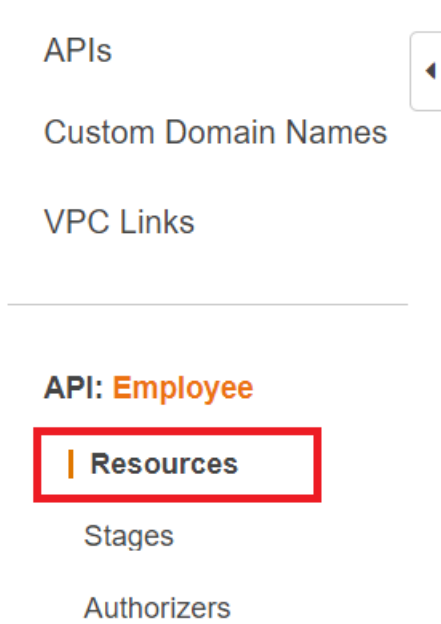
4. 직원을 API 이름으로 지정하고 설명을 입력합니다.

Settings

Choose a friendly name and description for your API.

API name*	Employee
Description	This invokes a Lambda function
Endpoint Type	Regional ⓘ

5. API 생성을 선택합니다.
6. 직원 섹션에서 리소스를 선택합니다.



7. 이름 필드에서 직원을 지정합니다.

8. 리소스 생성을 선택합니다.

9. 작업 드롭다운에서 리소스 생성을 선택합니다.

Use this page to create a new child resource for your resource. 

Configure as [proxy resource](#) ☐ 

Resource Name* employees

Resource Path* / employees

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/ {proxy+}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/foo`. To handle requests to `/`, add a new ANY method on the `/` resource.

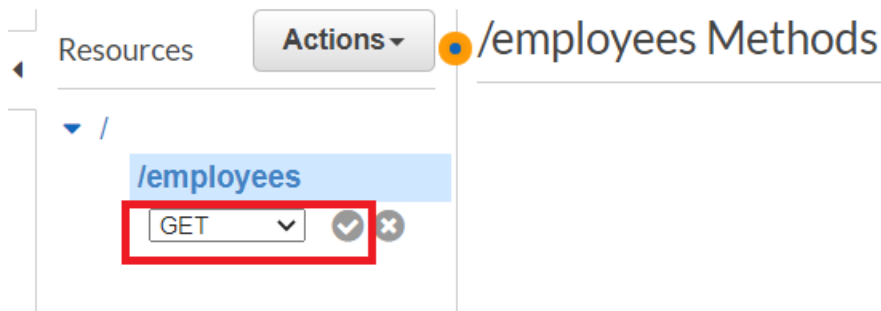
Enable API Gateway CORS ☐ 

* Required

Cancel

Create Resource

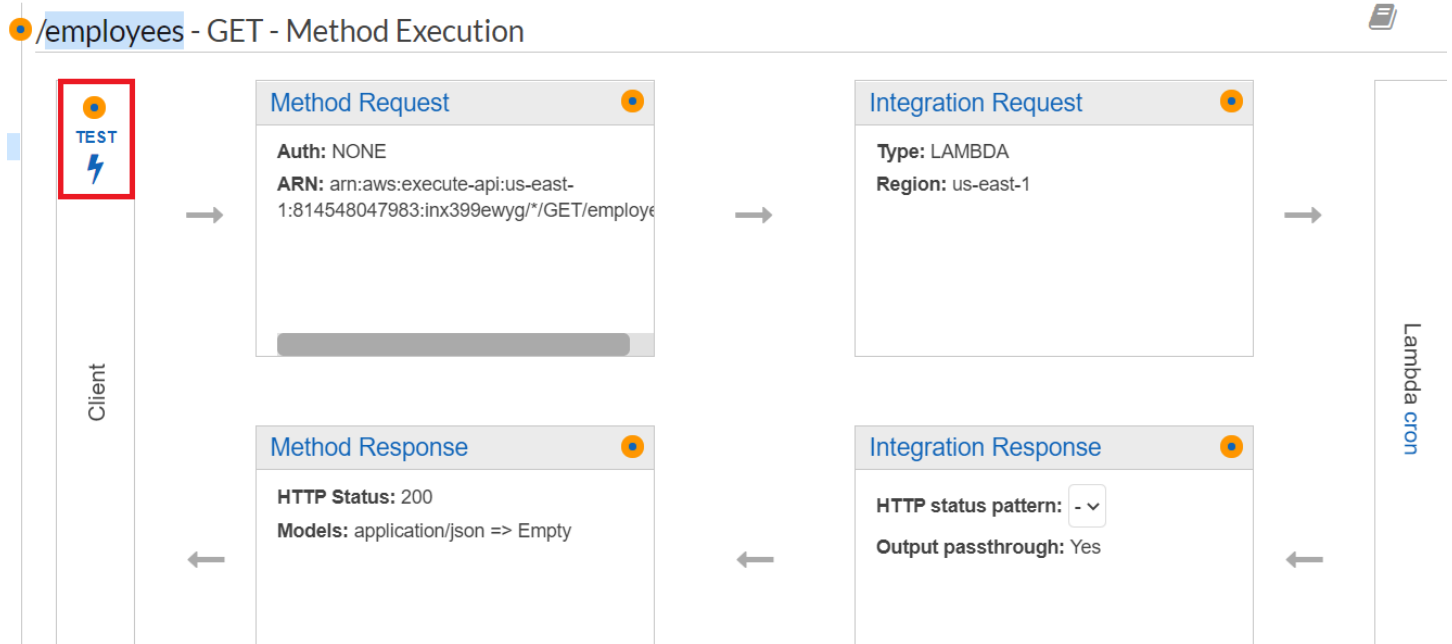
10. `/employees`를 선택하고, 작업에서 메서드 생성을 선택한 다음, `/employees` 아래의 드롭다운 메뉴에서 GET을 선택합니다. 체크 표시 아이콘을 선택합니다.



11. Lambda 함수를 선택하고 Lambda 함수 이름으로 `mylambdafunction`을 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.

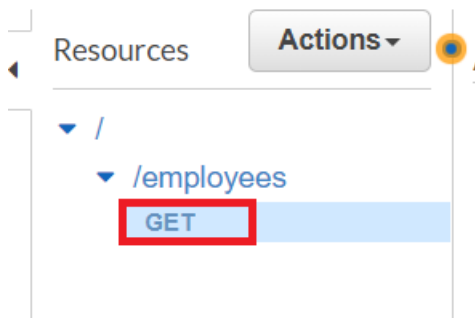


Lambda 함수가 간접적으로 호출되면 로그 파일을 보고 성공 메시지를 확인할 수 있습니다.

API Gateway 메서드 배포

테스트가 성공하면 [Amazon API Gateway 콘솔](#)에서 메서드를 배포할 수 있습니다.

1. GET을 선택합니다.



2. 작업 드롭다운에서 API 배포를 선택합니다.

Deploy API 

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	[New Stage]
Stage name*	lambdastage
Stage description	
Deployment description	

Cancel

3. API 배포 양식을 작성하고 배포를 선택합니다.

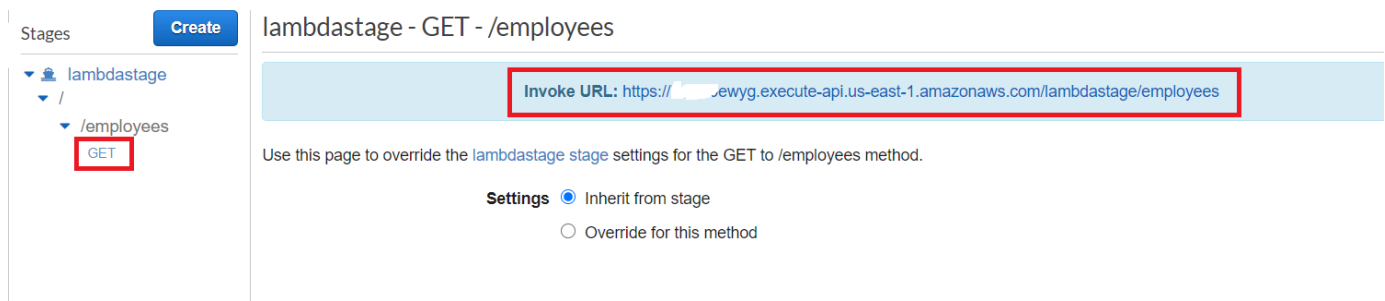
Deploy API 

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	[New Stage]
Stage name*	lambdastage
Stage description	
Deployment description	

Cancel

4. 변경 사항 저장(Save Changes)을 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 CloudFormation 스택을 삭제하여 이 작업을 수행할 수 있습니다.

1. [CloudFormationAWS 관리 콘솔](#)에서 엽니다.
2. 스택 페이지를 열고 스택을 선택합니다.
3. 삭제를 선택합니다.

AWS Lambda 함수를 실행하기 위한 예약된 이벤트 생성

Amazon CloudWatch 이벤트를 사용하여 AWS Lambda 함수를 호출하는 예약된 이벤트를 생성할 수 있습니다. cron 표현식을 사용하여 Lambda 함수가 간접적으로 호출되는 시기를 예약하도록 CloudWatch 이벤트를 구성할 수 있습니다. 예를 들어 평일마다 Lambda 함수를 간접적으로 호출하도록 CloudWatch 이벤트를 예약할 수 있습니다.

AWS Lambda 는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있는 컴퓨팅 서비스입니다. 다양한 프로그래밍 언어로 Lambda 함수를 생성할 수 있습니다. 에 대한 자세한 내용은 [란 AWS Lambda 무엇입니까?](#)를 AWS Lambda참조하십시오.

이 자습서에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 다양한 AWS 서비스를 호출하여 특정 사용 사례를 수행합니다. 예를 들어 다음 그림과 같이 조직에서 1주년을 맞이하는 직원들에게 축하하는 모바일 문자 메시지를 보낸다고 가정해 보겠습니다.

Today 2:50 PM

Malcolm happy one year anniversary. We are very happy that you have been working here for a year!

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 확인하는 방법, 데이터를 처리하고 문자 메시지를 보내는 방법을 모두 알아봅니다. 그런 다음, cron 표현식을 사용하여 평일마다 Lambda 함수를 간접적으로 호출하는 방법을 알아봅니다.

이 AWS 자습서에서는 이러한 필드가 포함된 Employee이라는 Amazon DynamoDB 테이블을 사용합니다.

- id - 테이블의 프라이머리 키입니다.
- firstName - 직원의 이름입니다.
- phone - 직원의 전화번호입니다.
- startDate - 직원의 시작 날짜입니다.

Scan ▾ [Table] Employee: Id ▾ ⬆ ⊕ Add filter Start search					
☐	Id ⓘ ▲	first ▼	phone ▼	startDate ▼	
☐	1	Scott	15555555654	2019-12-20	
☐	2	Malcolm	15555555654	2019-12-17	
☐	55	Lam	15555555654	2019-12-19	

⚠ 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 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 Id라는 키가 있는 Employee라는 Amazon DynamoDB 테이블. 이 사용 사례를 테스트하려는 유효한 휴대폰을 포함해 올바른 데이터를 입력했는지 확인하세요. 자세한 내용은 [테이블 생성](#)을 참조하세요.
- Lambda 함수를 실행하기 위한 권한이 연결된 IAM 역할.
- Lambda 함수를 호스팅하는 Amazon S3 버킷.

이러한 리소스를 수동으로 생성할 수 있지만 이 자습서에 설명된 CloudFormation 대로를 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

를 사용하여 AWS 리소스 생성 CloudFormation

CloudFormation 를 사용하면 AWS 인프라 배포를 예측 가능하고 반복적으로 생성하고 프로비저닝할 수 있습니다. 에 대한 자세한 내용은 [AWS CloudFormation 사용 설명서](#)를 CloudFormation참조하세요.

를 사용하여 CloudFormation 스택을 생성하려면 AWS CLI:

1. [AWS CLI 사용 설명서](#)의 AWS CLI 지침에 따라를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 디렉터리에 이름이 setup.yaml인 파일을 생성하고 [여기 GitHub의](#) 내용을 해당 파일에 복사합니다.

 Note

CloudFormation 템플릿은 [GitHub에서 사용할](#) 수 있는 AWS CDK 사용하여 생성되었습니다. 에 대한 자세한 내용은 [AWS Cloud Development Kit \(AWS CDK\) 개발자 안내서](#)를 AWS CDK참조하세요.

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 [CloudFormation 사용 설명서](#)를 참조하세요.

CloudFormation 대시보드에서 스택을 열고 리소스 탭을 선택하여 콘솔의 리소스 목록을 봅니다. 자습서에는 이러한 정보가 필요합니다.

4. 스택이 생성되면에 설명된 대로 AWS SDK for JavaScript 를 사용하여 DynamoDB 테이블을 채웁니다 [DynamoDB 테이블 채우기](#).

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 [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.
const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
    ],
  },
};
const command = new BatchWriteItemCommand(params);
dynamoClient.batchWriteItem(command).promise().then(() => {
  console.log("Batch write successful");
}, (err) => {
  console.error("Batch write failed", err);
});
```

```

        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

먼저 필수 AWS SDK for JavaScript (v3) 모듈 및 명령과 DynamoDBClient DynamoDB ScanCommand SNSClient 및 Amazon SNS PublishCommand 명령을 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 PublishCommand를 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 주제에서는 mylambdafunction.js 및 이 예제에 필요한 AWS SDK for JavaScript 모듈을 라는 번들 파일로 번들링하는 방법을 설명합니다 index.js.

1. webpack을 아직 설치하지 않았다면 이 예의 [사전 필수 작업](#)에 따라 설치합니다.

Note

Webpack에 관한 자세한 내용은 [Webpack으로 애플리케이션 번들링](#) 단원을 참조하세요.

2. 명령줄에서 다음을 실행하여 이 예의 JavaScript를 <index.js>라는 파일로 번들링합니다.

```

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.js`에 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_NAME을 Lambda 함수의 ZIP 버전을 업로드한 Amazon S3 버킷 이름으로 바꿉니다.
ZIP_FILE_NAME을 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_NAME을 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 예는 [여기 GitHub에서](#) 제공합니다.

Lambda 함수를 간접적으로 호출하도록 CloudWatch 구성

Lambda 함수를 간접적으로 호출하도록 CloudWatch를 구성하려면 다음을 수행합니다.

1. Lambda 콘솔에서 함수 페이지를 엽니다.
2. Lambda 함수를 선택합니다.
3. Designer에서 트리거 추가를 선택합니다.
4. 트리거 유형을 CloudWatch Events/EventBridge로 설정합니다.
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 [Amazon EventBridge에 AWS Lambda 사용\(CloudWatch Events\)](#) 단원을 참조하세요.

리소스 삭제

축하합니다! AWS SDK for JavaScript를 사용하여 Amazon CloudWatch 예약 이벤트를 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 CloudFormation 스택을 삭제하여이 작업을 수행할 수 있습니다.

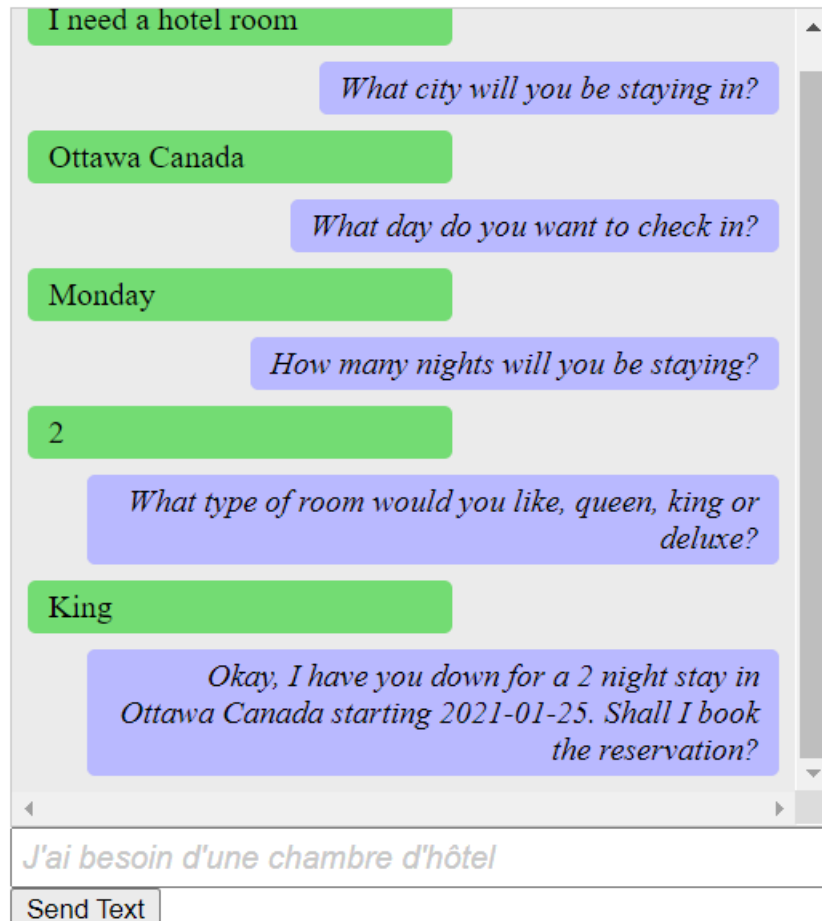
1. [CloudFormation 콘솔](#)을 엽니다.
2. 스택 페이지에서 스택을 선택합니다.
3. 삭제를 선택합니다.

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.

Amazon Lex - BookTrip

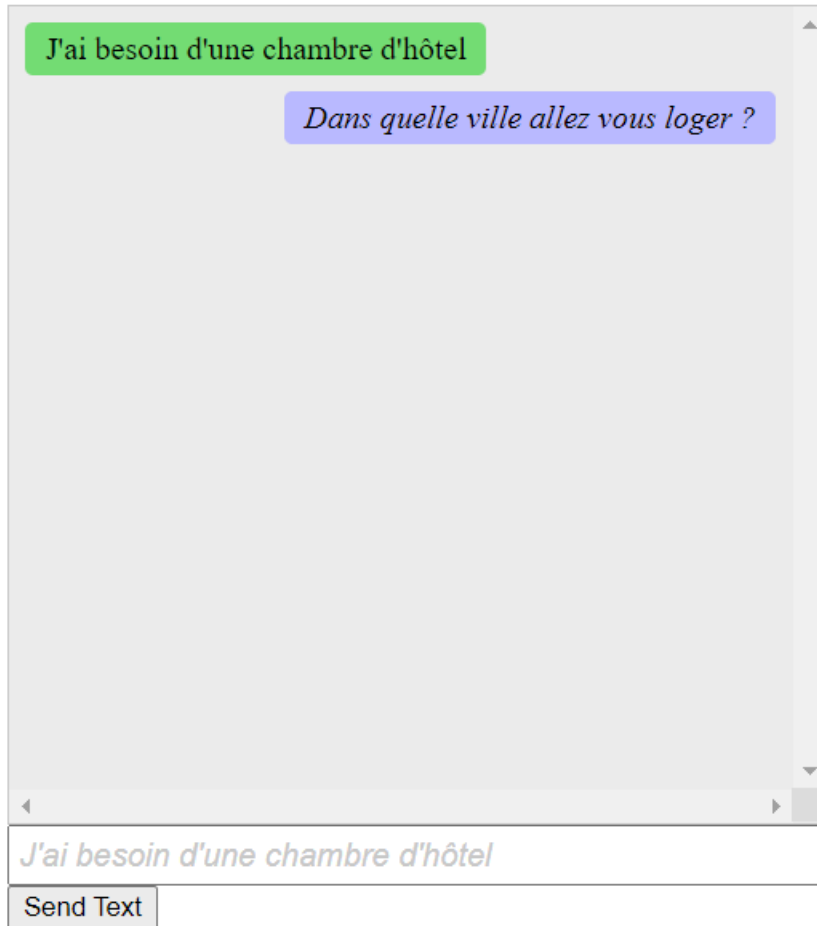
This multiple language chatbot shows you how easy it is to incorporate [Amazon Lex](#) into your web apps. Try it out.



이 AWS 자습서에서 생성된 Amazon Lex 챗봇은 여러 언어를 처리할 수 있습니다. 예를 들어 프랑스어를 구사하는 사용자는 프랑스어 텍스트를 입력하고 프랑스어로 응답을 받을 수 있습니다.

Amazon Lex - BookTrip

This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



마찬가지로, 사용자는 이탈리아어로 Amazon Lex 챗봇과 소통할 수 있습니다.

Amazon Lex - BookTrip

This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



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 티어](#)에 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](#)

사전 조건

이 예제를 설정하고 실행하려면 먼저 이러한 작업들을 완료해야 합니다.

- 이러한 노드 TypeScript 예제를 실행하도록 프로젝트 환경을 설정하고 필요한 AWS SDK for JavaScript 모듈과 타사 모듈을 설치합니다. [GitHub](#)의 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 downloads](#)를 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 AWS CloudFormation 대로를 사용하여 이러한 리소스를 프로비저닝하는 것이 좋습니다.

를 사용하여 AWS 리소스 생성 CloudFormation

CloudFormation 를 사용하면 AWS 인프라 배포를 예측 가능하고 반복적으로 생성하고 프로비저닝할 수 있습니다. 에 대한 자세한 내용은 [AWS CloudFormation 사용 설명서](#)를 CloudFormation참조하세요.

를 사용하여 CloudFormation 스택을 생성하려면 AWS CLI:

1. [AWS CLI 사용 설명서](#)의 AWS CLI 지침에 따라를 설치하고 구성합니다.
2. 프로젝트 폴더의 루트 디렉터리에 이름이 `setup.yaml`인 파일을 생성하고 [여기 GitHub의](#) 내용을 해당 파일에 복사합니다.

 Note

CloudFormation 템플릿은 [GitHub에서 사용할](#) 수 있는 AWS CDK 사용하여 생성되었습니다. 에 대한 자세한 내용은 [AWS Cloud Development Kit \(AWS CDK\) 개발자 안내서](#)를 AWS CDK참조하세요.

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 [CloudFormation 사용 설명서](#)를 참조하세요.

생성된 리소스를 보려면 Amazon Lex 콘솔을 열고 스택을 선택한 다음, 리소스 탭을 선택합니다.

Amazon Lex 봇 생성

 Important

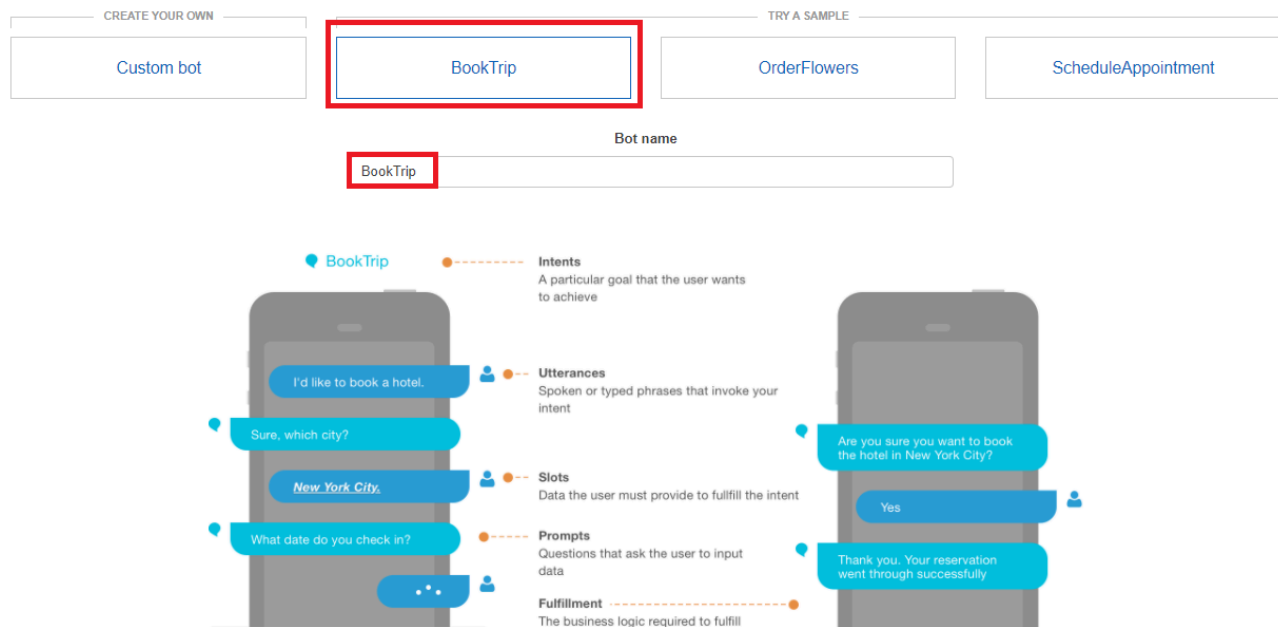
Amazon Lex 콘솔 V1을 사용하여 봇을 생성합니다. 이 예는 V2를 사용하여 생성된 봇에서는 작동하지 않습니다.

첫 번째 단계는 Amazon Web Services Management Console을 사용하여 Amazon Lex 챗봇을 생성하는 것입니다. 이 예에서는 Amazon Lex BookTrip 예가 사용됩니다. 자세한 내용은 [Book Trip](#) 단원을 참조하세요.

- Amazon Web Services Management Console에 로그인하고 [Amazon Web Services 콘솔](#)에서 Amazon Lex 콘솔을 엽니다.
- 봇 페이지에서 생성을 선택합니다.
- BookTrip 청사진을 선택합니다(기본 봇 이름인 BookTrip을 그대로 유지).

Create your bot

Amazon Lex enables any developer to build conversational chatbots quickly and easily. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversational flow directly from the console, and then Amazon Lex manages the dialogue and dynamically adjusts the response. To get started, you can choose one of the sample bots provided below or build a new custom bot from scratch.



- 기본 설정을 입력하고 생성을 선택합니다(콘솔에 BookTrip 봇이 표시됨). 편집기 탭에서 미리 구성된 intent의 세부 정보를 검토합니다.
- 테스트 창에서 봇을 테스트합니다. 호텔 객실을 예약하고 싶어요를 입력해 테스트를 시작합니다.

> Test bot (Latest)

✔ Ready. Build complete

I want to book a hotel room

What city will you be staying in?

Clear chat history

 Chat with your bot...

Inspect response

Dialog State: ElicitSlot

Hide

☒ Summary ☐ Detail

Intent: BookHotel

- 게시를 선택하고 별칭 이름을 지정합니다(사용 시이 값이 필요함 AWS SDK for JavaScript).

 Note

JavaScript 코드에서 봇 이름과 봇 별칭을 참조해야 합니다.

HTML 생성

index.html이라는 이름의 파일을 만듭니다. 아래 코드를 복사하여 index.html에 붙여 넣습니다. 이 HTML은 main.js를 참조합니다. 이는 필수 AWS SDK for JavaScript 모듈을 포함하는 index.js의 번들 버전입니다. [HTML 생성](#)에서 이 파일을 생성합니다. 또한 index.html은 스타일을 추가하는 style.css도 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 [여기 GitHub에서](#)도 제공합니다.

브라우저 스크립트 생성

index.js이라는 이름의 파일을 만듭니다. 아래 코드를 복사하여 index.js에 붙여 넣습니다. 필요한 AWS SDK for JavaScript 모듈과 명령을 가져옵니다. Amazon Lex, Amazon Comprehend 및 Amazon Translate용 클라이언트를 생성합니다. **REGION**을 AWS 리전으로 바꾸고 **IDENTITY_POOL_ID**를에서 생성한 자격 증명 풀의 ID로 바꿉니다. [AWS 리소스 생성](#). 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.js라는 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

를 사용하여 [AWS 리소스 생성 CloudFormation](#)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 { 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 [여기 GitHub에서](#) 제공합니다.

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";
```



```
let g_text = "";
// Set the focus to the input box.
document.getElementById("wisdom").focus();

function showRequest() {
  const conversationDiv = document.getElementById("conversation");
  const requestPara = document.createElement("P");
  requestPara.className = "userRequest";
  requestPara.appendChild(document.createTextNode(g_text));
  conversationDiv.appendChild(requestPara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function showResponse(lexResponse) {
  const conversationDiv = document.getElementById("conversation");
  const responsePara = document.createElement("P");
  responsePara.className = "lexResponse";

  const 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;
  const 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.
  const wisdomText = document.getElementById("wisdom");
  wisdomText.value = "";
  wisdomText.locked = false;
}
```

```
}

// Respond to user's input.
const createResponse = async () => {
  // Confirm there is text to submit.
  const wisdomText = document.getElementById("wisdom");
  if (wisdomText?.value && wisdomText.value.trim().length > 0) {
    // Disable input to show it is being sent.
    const 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'.
        };
      } catch {
        try {
          const data = await lexClient.send(new PostTextCommand(lexParams));
          console.log("Success. Response is: ", data.message);
          const 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 [여기 GitHub에서](#) 제공합니다.

이제 Webpack을 사용하여 index.js 및 AWS SDK for JavaScript 모듈을 단일 파일로 번들링합니다. main.js.

1. webpack을 아직 설치하지 않았다면 이 예의 [사전 조건](#)에 따라 설치합니다.

Note

Webpack에 관한 자세한 내용은 [Webpack으로 애플리케이션 번들링](#) 단원을 참조하세요.

2. 명령줄에서 다음을 실행하여 이 예의 JavaScript를 main.js라는 파일로 번들링합니다.

```
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 CloudFormation 스택을 삭제하여 이 작업을 수행할 수 있습니다.

1. [CloudFormation 콘솔](#)을 엽니다.
2. 스택 페이지에서 스택을 선택합니다.
3. 삭제를 선택합니다.

더 많은 AWS 교차 서비스 예제는 [AWS SDK for JavaScript 교차 서비스 예제를 참조하세요](#).

SDK for JavaScript(v3) 코드 예

이 주제의 코드 예제에서는 AWS SDK for JavaScript (v3)를와 함께 사용하는 방법을 보여줍니다 AWS.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

일부 서비스에는 서비스와 관련된 라이브러리 또는 함수를 활용하는 방법을 보여주는 추가 예제 범주가 포함되어 있습니다.

서비스

- [SDK for JavaScript\(v3\)를 사용한 API Gateway 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Aurora 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Auto Scaling 예시](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Bedrock 예시](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Bedrock Runtime 예시](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Bedrock Agents 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Bedrock Agents Runtime 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 CloudWatch 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 CloudWatch Events 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 CloudWatch Logs 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 CodeBuild 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Cognito ID 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon Cognito 자격 증명 공급자 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Comprehend 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon DocumentDB 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 DynamoDB 예](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon EC2 예](#)

- [SDK for JavaScript\(v3\)를 사용한 Elastic Load Balancing 버전 2 예제](#)
- [AWS Entity Resolution SDK for JavaScript\(v3\)를 사용한 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 EventBridge 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Glacier 예제](#)
- [AWS Glue SDK for JavaScript\(v3\)를 사용한 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 HealthImaging 예](#)
- [SDK for JavaScript \(v3\)를 사용한 IAM 예제](#)
- [AWS IoT SiteWise SDK for JavaScript\(v3\)를 사용한 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Kinesis 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 Lambda 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Lex 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Location 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon MSK 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon Personalize 예](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon Personalize 이벤트 예](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon Personalize 런타임 예](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon Pinpoint 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Polly 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon RDS 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon RDS 데이터 서비스 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon Redshift 예](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Rekognition 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon S3 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 SageMaker AI 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 Secrets Manager 예](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon SES 예](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon SNS 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon SQS 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Step Functions의 예시](#)
- [AWS STS SDK for JavaScript\(v3\)를 사용한 예제](#)

- [지원 SDK for JavaScript\(v3\)를 사용한 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Systems Manager 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Textract 예제](#)
- [SDK for JavaScript \(v3\)를 사용한 Amazon Transcribe 예제](#)
- [SDK for JavaScript\(v3\)를 사용한 Amazon Translate 예제](#)

SDK for JavaScript(v3)를 사용한 API Gateway 예제

다음 코드 예제에서는 API Gateway와 함께 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

사진을 관리하기 위한 서버리스 애플리케이션 만들기

다음 코드 예시에서는 사용자가 레이블을 사용하여 사진을 관리할 수 있는 서버리스 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 [AWS 커뮤니티](#)의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

API Gateway를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon API Gateway에서 호출한 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

Lambda JavaScript 런타임 API를 사용하여 AWS Lambda 함수를 생성하는 방법을 보여줍니다. 이 예제에서는 다양한 AWS 서비스를 호출하여 특정 사용 사례를 수행합니다. 이 예제에서는 Amazon API Gateway에서 간접 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

SDK for JavaScript(v3)를 사용한 Aurora 예제

다음 코드 예제에서는 Aurora에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

Aurora 서버리스 작업 항목 트래커 만들기

다음 코드 예제에서는 Amazon Aurora Serverless 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 사용하여 보고서를 보내는 웹 애플리케이션을 만드는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 보여줍니다. 이 예제에서는 Express Node.js 백엔드와의 상호 작용을 위해 React.js로 빌드된 프론트엔드를 사용합니다.

- React.js 웹 애플리케이션을와 통합합니다 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

SDK for JavaScript(v3)를 사용한 Auto Scaling 예시

다음 코드 예제에서는 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)
- [시나리오](#)

작업

AttachLoadBalancerTargetGroups

다음 코드 예시는 AttachLoadBalancerTargetGroups의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [AttachLoadBalancerTargetGroups](#)를 참조하세요.

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.
- Auto Scaling 그룹의 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 파라미터를 업데이트하여 요청 및 상태 확인에 대한 웹 서버 응답을 제어합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}

```

모든 리소스를 배포하기 위한 단계를 생성합니다.

```
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 { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 () => {
    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,
        },
      }),
    );
  })),
  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) => {
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {

```

```

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,
  }),
);
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) => {
  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] },
  );
});

```

```

    }),
    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) => {
      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 },
          ],
        })
      );
      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) => {
      const client = new AutoScalingClient({});
      await client.send(
        new AttachLoadBalancerTargetGroupsCommand({
          AutoScalingGroupName: NAMES.autoScalingGroupName,

```

```

        TargetGroupARNs: [state.targetGroupArn],
    })),
    );
  })),
  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),
        );
      }
      return MESSAGES.noIpRules;
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      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);
  }
  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,
    );
  }
}),
saveState,
];

```

데모를 실행하기 위한 단계를 생성합니다.

```
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) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  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: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      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];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );

      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",
    }),
  );
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
  }),
);

return InstanceProfile;
}

```

모든 리소스를 폐기하는 단계를 생성합니다.

```

import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} 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 { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 = [
    loadState,
    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(
          "${TABLE_NAME}",
          NAMES.tableName,
        );
      }
      return MESSAGES.deletedTable.replace("${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,
        );
      }
      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.`,
          );
        }
      }
    })
  ],
  {
    state: {
      deleteTableError: null,
      deleteKeyPairError: null,
      detachPolicyFromRoleError: null,
    },
  },
);

```

```

    } 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);
  }
  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,
    );
}
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("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.removedRoleFromInstanceProfile
        .replace("${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(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
  return MESSAGES.deletedInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  );
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
  return MESSAGES.deletedInstanceProfile.replace(
    "${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(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  })),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  })),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  })),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {

```

```

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.");
  }
});
} catch (e) {
  state.deleteLoadBalancerError = e;
}
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
  return MESSAGES.deletedLoadBalancer.replace(
    "${LB_NAME}",
    NAMES.loadBalancerName,
  );
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  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;
}
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }
  return MESSAGES.deletedLoadBalancerTargetGroup.replace(
    "${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);
  }
  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);
    }
    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");
    }
  })

```

```

    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(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${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(
            "${POLICY_NAME}",
            NAMES.ssmOnlyPolicyName,
        );
    }
    return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${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(
            "${ROLE_NAME}",
            NAMES.ssmOnlyRoleName,
        );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
    );
}),
new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
        /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
        state,
    ) => {
        const ec2Client = new EC2Client({});

```



```

    try {
      await ec2Client.send(
        new RevokeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    } catch (e) {
      state.revokeSecurityGroupIngressError = e;
    }
  },
),
new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
  if (state.revokeSecurityGroupIngressError) {
    console.error(state.revokeSecurityGroupIngressError);
    return MESSAGES.revokeSecurityGroupIngressError.replace(
      "${IP}",
      state.myIp,
    );
  }
  return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
}),
];

/**
 * @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;
    }
    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](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

SDK for JavaScript(v3)를 사용한 Amazon Bedrock 예시

다음 코드 예제에서는 Amazon Bedrock에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)

시작하기

Amazon Bedrock 시작

다음 코드 예제에서는 Amazon Bedrock 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node: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 (const 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 참조의 [ListFoundationModels](#)를 참조하세요.

작업

GetFoundationModel

다음 코드 예시는 GetFoundationModel의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

파운데이션 모델에 대한 세부 정보를 가져옵니다.

```

import { fileURLToPath } from "node: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 참조의 [GetFoundationModel](#)을 참조하세요.

ListFoundationModels

다음 코드 예시는 ListFoundationModels의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배우보세요.

사용 가능한 파운데이션 모델을 나열합니다.

```

import { fileURLToPath } from "node: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 참조의 [ListFoundationModels](#)를 참조하세요.

SDK for JavaScript(v3)를 사용한 Amazon Bedrock Runtime 예시

다음 코드 예제에서는 Amazon Bedrock 런타임과 함께 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [시나리오](#)
- [Amazon Nova](#)
- [Amazon Nova Canvas](#)
- [Anthropic Claude](#)
- [Cohere Command](#)
- [Meta Llama](#)
- [Mistral AI](#)

시작하기

Amazon Bedrock 시작

다음 코드 예제에서는 Amazon Bedrock 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 * @typedef {Object} Content
 * @property {string} text
 *
 * @typedef {Object} Usage
 * @property {number} input_tokens
 * @property {number} output_tokens
 *
 * @typedef {Object} ResponseBody
 * @property {Content[]} content
 * @property {Usage} usage
 */
```

```
import { fileURLToPath } from "node:url";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

const AWS_REGION = "us-east-1";

const MODEL_ID = "anthropic.claude-3-haiku-20240307-v1:0";
const PROMPT = "Hi. In a short paragraph, explain what you can do.";

const hello = async () => {
  console.log("=".repeat(35));
  console.log("Welcome to the Amazon Bedrock demo!");
  console.log("=".repeat(35));

  console.log("Model: Anthropic Claude 3 Haiku");
  console.log(`Prompt: ${PROMPT}\n`);
  console.log("Invoking model...\n");

  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: AWS_REGION });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [{ role: "user", content: [{ type: "text", text: PROMPT }] }],
  };

  // Invoke Claude with the payload and wait for the response.
  const apiResponse = await client.send(
    new InvokeModelCommand({
      contentType: "application/json",
      body: JSON.stringify(payload),
      modelId: MODEL_ID,
    })
  );

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  const responses = responseBody.content;
```

```

if (responses.length === 1) {
  console.log(`Response: ${responses[0].text}`);
} else {
  console.log("Haiku returned multiple responses:");
  console.log(responses);
}

console.log(`\nNumber of input tokens:  ${responseBody.usage.input_tokens}`);
console.log(`Number of output tokens: ${responseBody.usage.output_tokens}`);
};

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await hello();
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [InvokeModel](#)을 참조하세요.

시나리오

Amazon Bedrock에서 여러 파운데이션 모델 간접 호출

다음 코드 예제에서는 Amazon Bedrock에서 프롬프트를 준비하고 다양한 대규모 언어 모델(LLM)에 전송하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { fileURLToPath } from "node:url";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,

```

```

} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { FoundationModels } from "../config/foundation_models.js";

/**
 * @typedef {Object} ModelConfig
 * @property {Function} module
 * @property {Function} invoker
 * @property {string} modelId
 * @property {string} modelName
 */

const greeting = new ScenarioOutput(
  "greeting",
  "Welcome to the Amazon Bedrock Runtime client demo!",
  { header: true },
);

const selectModel = new ScenarioInput("model", "First, select a model:", {
  type: "select",
  choices: Object.values(FoundationModels).map((model) => ({
    name: model.modelName,
    value: model,
  })),
});

const enterPrompt = new ScenarioInput("prompt", "Now, enter your prompt:", {
  type: "input",
});

const printDetails = new ScenarioOutput(
  "print details",
  /**
   * @param {{ model: ModelConfig, prompt: string }} c
   */
  (c) => console.log(`Invoking ${c.model.modelName} with '${c.prompt}'...`),
);

const invokeModel = new ScenarioAction(
  "invoke model",
  /**
   * @param {{ model: ModelConfig, prompt: string, response: string }} c
   */
  async (c) => {
    const modelModule = await c.model.module();

```

```

    const invoker = c.model.invoker(modelModule);
    c.response = await invoker(c.prompt, c.model.modelId);
  },
);

const printResponse = new ScenarioOutput(
  "print response",
  /**
   * @param {{ response: string }} c
   */
  (c) => c.response,
);

const scenario = new Scenario("Amazon Bedrock Runtime Demo", [
  greeting,
  selectModel,
  enterPrompt,
  printDetails,
  invokeModel,
  printResponse,
]);

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  scenario.run();
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [InvokeModel](#)
 - [InvokeModelWithResponseStream](#)

Converse API에서 도구 사용

다음 코드 예제에서는 애플리케이션, 생성형 AI 모델, 연결된 도구 또는 API 간에 일반적인 상호 작용을 구축하여 AI와 외부 환경 간의 상호 작용을 매개하는 방법을 보여줍니다. 외부 날씨 API를 AI 모델에 연결하는 예제를 사용하면 사용자 입력에 따라 실시간 날씨 정보를 제공할 수 있습니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

시나리오 흐름의 기본 실행입니다. 이 시나리오는 사용자, Amazon Bedrock Converse API 및 날씨 도구 간의 대화를 오케스트레이션합니다.

```
/* Before running this JavaScript code example, set up your development environment,
   including your credentials.
   This demo illustrates a tool use scenario using Amazon Bedrock's Converse API and a
   weather tool.
   The script interacts with a foundation model on Amazon Bedrock to provide weather
   information based on user
   input. It uses the Open-Meteo API (https://open-meteo.com) to retrieve current
   weather data for a given location.*/

import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";
import data from "./questions.json" with { type: "json" };
import toolConfig from "./tool_config.json" with { type: "json" };

const __filename = fileURLToPath(import.meta.url);

const systemPrompt = [
  {
    text:
```

```

        "You are a weather assistant that provides current weather data for user-
specified locations using only\n" +
        "the Weather_Tool, which expects latitude and longitude. Infer the coordinates
from the location yourself.\n" +
        "If the user provides coordinates, infer the approximate location and refer to
it in your response.\n" +
        "To use the tool, you strictly apply the provided tool specification.\n" +
        "If the user specifies a state, country, or region, infer the locations of
cities within that state.\n" +
        "\n" +
        "- Explain your step-by-step process, and give brief updates before each step.
\n" +
        "- Only use the Weather_Tool for data. Never guess or make up information. \n"
+
        "- Repeat the tool use for subsequent requests if necessary.\n" +
        "- If the tool errors, apologize, explain weather is unavailable, and suggest
other options.\n" +
        "- Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports
concise. Sparingly use\n" +
        " emojis where appropriate.\n" +
        "- Only respond to weather queries. Remind off-topic users of your purpose.
\n" +
        "- Never claim to search online, access external data, or use tools besides
Weather_Tool.\n" +
        "- Complete the entire process until you have all required data before sending
the complete response.",
    },
];
const tools_config = toolConfig;

/// Starts the conversation with the user and handles the interaction with Bedrock.
async function askQuestion(userMessage) {
    // The maximum number of recursive calls allowed in the tool use function.
    // This helps prevent infinite loops and potential performance issues.
    const max_recursions = 5;
    const messages = [
        {
            role: "user",
            content: [{ text: userMessage }],
        },
    ];
    try {
        const response = await SendConversationtoBedrock(messages);
        await ProcessModelResponseAsync(response, messages, max_recursions);
    }
}

```

```

    } catch (error) {
        console.log("error ", error);
    }
}

// Sends the conversation, the system prompt, and the tool spec to Amazon Bedrock,
// and returns the response.
// param "messages" - The conversation history including the next message to send.
// return - The response from Amazon Bedrock.
async function SendConversationtoBedrock(messages) {
    const bedRockRuntimeClient = new BedrockRuntimeClient({
        region: "us-east-1",
    });
    try {
        const modelId = "amazon.nova-lite-v1:0";
        const response = await bedRockRuntimeClient.send(
            new ConverseCommand({
                modelId: modelId,
                messages: messages,
                system: systemPrompt,
                toolConfig: tools_config,
            }),
        );
        return response;
    } catch (caught) {
        if (caught.name === "ModelNotReady") {
            console.log(
                ``${caught.name}` - Model not ready, please wait and try again.",
            );
            throw caught;
        }
        if (caught.name === "BedrockRuntimeException") {
            console.log(
                ``${caught.name}` - "Error occurred while sending Converse request.",
            );
            throw caught;
        }
    }
}

// Processes the response received via Amazon Bedrock and performs the necessary
// actions based on the stop reason.
// param "response" - The model's response returned via Amazon Bedrock.
// param "messages" - The conversation history.

```



```
// param "max_recursions" - The maximum number of recursive calls allowed.
async function ProcessModelResponseAsync(response, messages, max_recursions) {
  if (max_recursions <= 0) {
    await HandleToolUseAsync(response, messages);
  }
  if (response.stopReason === "tool_use") {
    await HandleToolUseAsync(response, messages, max_recursions - 1);
  }
  if (response.stopReason === "end_turn") {
    const messageToPrint = response.output.message.content[0].text;
    console.log(messageToPrint.replace(/<[^>]+>/g, ""));
  }
}

// Handles the tool use case by invoking the specified tool and sending the tool's
// response back to Bedrock.
// The tool response is appended to the conversation, and the conversation is sent
// back to Amazon Bedrock for further processing.
// param "response" - the model's response containing the tool use request.
// param "messages" - the conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function HandleToolUseAsync(response, messages, max_recursions) {
  const toolResultFinal = [];
  try {
    const output_message = response.output.message;
    messages.push(output_message);
    const toolRequests = output_message.content;
    const toolMessage = toolRequests[0].text;
    console.log(toolMessage.replace(/<[^>]+>/g, ""));
    for (const toolRequest of toolRequests) {
      if (Object.hasOwn(toolRequest, "toolUse")) {
        const toolUse = toolRequest.toolUse;
        const latitude = toolUse.input.latitude;
        const longitude = toolUse.input.longitude;
        const toolUseID = toolUse.toolUseId;
        console.log(
          `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
        );
        if (toolUse.name === "Weather_Tool") {
          try {
            const current_weather = await callWeatherTool(
              longitude,
              latitude,
            ).then((current_weather) => current_weather);
            const currentWeather = current_weather;

```

```

        const toolResult = {
            toolResult: {
                toolUseId: toolUseID,
                content: [{ json: currentWeather }],
            },
        };
        toolResultFinal.push(toolResult);
    } catch (err) {
        console.log("An error occurred. ", err);
    }
}
}

const toolResultMessage = {
    role: "user",
    content: toolResultFinal,
};
messages.push(toolResultMessage);
// Send the conversation to Amazon Bedrock
await ProcessModelResponseAsync(
    await SendConversationtoBedrock(messages),
    messages,
);
} catch (error) {
    console.log("An error occurred. ", error);
}
}
// Call the Weathertool.
// param = longitude of location
// param = latitude of location
async function callWeatherTool(longitude, latitude) {
    // Open-Meteo API endpoint
    const apiUrl = `https://api.open-meteo.com/v1/forecast?latitude=${latitude}&longitude=${longitude}&current_weather=true`;

    // Fetch the weather data.
    return fetch(apiUrl)
        .then((response) => {
            return response.json().then((current_weather) => {
                return current_weather;
            });
        })
        .catch((error) => {

```

```

        console.error("Error fetching weather data:", error);
    });
}
/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
    type: "input",
    default: "",
});

const greet = new ScenarioOutput(
    "greet",
    "Welcome to the Amazon Bedrock Tool Use demo! \n" +
    "This assistant provides current weather information for user-specified locations. " +
    "You can ask for weather details by providing the location name or coordinates." +
    "Weather information will be provided using a custom Tool and open-meteo API." +
    "For the purposes of this example, we'll use in order the questions in ./questions.json :\n" +
    "What's the weather like in Seattle? " +
    "What's the best kind of cat? " +
    "Where is the warmest city in Washington State right now? " +
    "What's the warmest city in California right now?\n" +
    "To exit the program, simply type 'x' and press Enter.\n" +
    "Have fun and experiment with the app by editing the questions in ./questions.json! " +
    "P.S.: You're not limited to single locations, or even to using English! ",

    { header: true },
);

const displayAskQuestion1 = new ScenarioOutput(
    "displayAskQuestion1",
    "Press enter to ask question number 1 (default is 'What's the weather like in Seattle?')",
);

const askQuestion1 = new ScenarioAction(
    "askQuestion1",
    async (** @type {State} */ state) => {
        const userMessage1 = data.questions["question-1"];
        await askQuestion(userMessage1);
    }
);

```

```
    },
  );

const displayAskQuestion2 = new ScenarioOutput(
  "displayAskQuestion2",
  "Press enter to ask question number 2 (default is 'What's the best kind of cat?')",
);

const askQuestion2 = new ScenarioAction(
  "askQuestion2",
  async (** @type {State} */ state) => {
    const userMessage2 = data.questions["question-2"];
    await askQuestion(userMessage2);
  },
);

const displayAskQuestion3 = new ScenarioOutput(
  "displayAskQuestion3",
  "Press enter to ask question number 3 (default is 'Where is the warmest city in Washington State right now?')",
);

const askQuestion3 = new ScenarioAction(
  "askQuestion3",
  async (** @type {State} */ state) => {
    const userMessage3 = data.questions["question-3"];
    await askQuestion(userMessage3);
  },
);

const displayAskQuestion4 = new ScenarioOutput(
  "displayAskQuestion4",
  "Press enter to ask question number 4 (default is 'What's the warmest city in California right now?')",
);

const askQuestion4 = new ScenarioAction(
  "askQuestion4",
  async (** @type {State} */ state) => {
    const userMessage4 = data.questions["question-4"];
    await askQuestion(userMessage4);
  },
);
```

```
const goodbye = new ScenarioOutput(
  "goodbye",
  "Thank you for checking out the Amazon Bedrock Tool Use demo. We hope you\n" +
    "learned something new, or got some inspiration for your own apps today!\n" +
    "For more Bedrock examples in different programming languages, have a look at:\n" +
    "https://docs.aws.amazon.com/bedrock/latest/userguide/service_code_examples.html",
);

const myScenario = new Scenario("Converse Tool Scenario", [
  greet,
  pressEnter,
  displayAskQuestion1,
  askQuestion1,
  pressEnter,
  displayAskQuestion2,
  askQuestion2,
  pressEnter,
  displayAskQuestion3,
  askQuestion3,
  pressEnter,
  displayAskQuestion4,
  askQuestion4,
  pressEnter,
  goodbye,
]);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

```
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Converse](#)를 참조하세요.

Amazon Nova

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Amazon Nova로 텍스트 메시지를 전송하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Amazon Nova로 텍스트 메시지를 전송합니다.

```
// This example demonstrates how to use the Amazon Nova foundation models to
// generate text.
// It shows how to:
// - Set up the Amazon Bedrock runtime client
// - Create a message
// - Configure and send a request
// - Process the response

import {
  BedrockRuntimeClient,
  ConversationRole,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Step 1: Create the Amazon Bedrock runtime client
// Credentials will be automatically loaded from the environment.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Step 2: Specify which model to use:
// Available Amazon Nova models and their characteristics:
```

```
// - Amazon Nova Micro: Text-only model optimized for lowest latency and cost
// - Amazon Nova Lite: Fast, low-cost multimodal model for image, video, and text
// - Amazon Nova Pro: Advanced multimodal model balancing accuracy, speed, and
  cost
//
// For the most current model IDs, see:
// https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
const modelId = "amazon.nova-lite-v1:0";

// Step 3: Create the message
// The message includes the text prompt and specifies that it comes from the user
const inputText =
  "Describe the purpose of a 'hello world' program in one line.";
const message = {
  content: [{ text: inputText }],
  role: ConversationRole.USER,
};

// Step 4: Configure the request
// Optional parameters to control the model's response:
// - maxTokens: maximum number of tokens to generate
// - temperature: randomness (max: 1.0, default: 0.7)
// OR
// - topP: diversity of word choice (max: 1.0, default: 0.9)
// Note: Use either temperature OR topP, but not both
const request = {
  modelId,
  messages: [message],
  inferenceConfig: {
    maxTokens: 500, // The maximum response length
    temperature: 0.5, // Using temperature for randomness control
    //topP: 0.9, // Alternative: use topP instead of temperature
  },
};

// Step 5: Send and process the request
// - Send the request to the model
// - Extract and return the generated text from the response
try {
  const response = await client.send(new ConverseCommand(request));
  console.log(response.output.message.content[0].text);
} catch (error) {
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  throw error;
}
```

```
}
```

도구 구성과 함께 Bedrock의 Converse API를 사용하여 Amazon Nova에 메시지 대화를 전송합니다.

```
// This example demonstrates how to send a conversation of messages to Amazon Nova
// using Bedrock's Converse API with a tool configuration.
// It shows how to:
// - 1. Set up the Amazon Bedrock runtime client
// - 2. Define the parameters required enable Amazon Bedrock to use a tool when
//   formulating its response (model ID, user input, system prompt, and the tool spec)
// - 3. Send the request to Amazon Bedrock, and returns the response.
// - 4. Add the tool response to the conversation, and send it back to Amazon
//   Bedrock.
// - 5. Publish the response.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Step 1: Create the Amazon Bedrock runtime client

// Credentials will be automatically loaded from the environment
const bedRockRuntimeClient = new BedrockRuntimeClient({
  region: "us-east-1",
});

// Step 2. Define the parameters required enable Amazon Bedrock to use a tool when
// formulating its response.

// The Bedrock Model ID.
const modelId = "amazon.nova-lite-v1:0";

// The system prompt to help Amazon Bedrock craft it's response.
const system_prompt = [
  {
    text:
      "You are a music expert that provides the most popular song played on a radio
      station, using only the\n" +
```



```

    "the top_song tool, which he call sign for the radio station for which you
    want the most popular song. " +
    "Example calls signs are WZPZ and WKRP. \n" +
    "- Only use the top_song tool. Never guess or make up information. \n" +
    "- If the tool errors, apologize, explain weather is unavailable, and suggest
    other options.\n" +
    "- Only respond to queries about the most popular song played on a radio
    station\n" +
    "Remind off-topic users of your purpose. \n" +
    "- Never claim to search online, access external data, or use tools besides
    the top_song tool.\n",
  },
];
// The user's question.
const message = [
  {
    role: "user",
    content: [{ text: "What is the most popular song on WZPZ?" }],
  },
];
// The tool specification. In this case, it uses an example schema for
// a tool that gets the most popular song played on a radio station.
const tool_config = {
  tools: [
    {
      toolSpec: {
        name: "top_song",
        description: "Get the most popular song played on a radio station.",
        inputSchema: {
          json: {
            type: "object",
            properties: {
              sign: {
                type: "string",
                description:
                  "The call sign for the radio station for which you want the most
                  popular song. Example calls signs are WZPZ and WKRP.",
              },
            },
            required: ["sign"],
          },
        },
      },
    },
  ],
},

```

```

    ],
  };

// Helper function to return the song and artist from top_song tool.
async function get_top_song(call_sign) {
  try {
    if (call_sign === "WZPZ") {
      const song = "Elemental Hotel";
      const artist = "8 Storey Hike";
      return { song, artist };
    }
  } catch (error) {
    console.log(`${error.message}`);
  }
}

// 3. Send the request to Amazon Bedrock, and returns the response.
export async function SendConversationtoBedrock(
  modelId,
  message,
  system_prompt,
  tool_config,
) {
  try {
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: message,
        system: system_prompt,
        toolConfig: tool_config,
      }),
    );
    if (response.stopReason === "tool_use") {
      const toolResultFinal = [];
      try {
        const output_message = response.output.message;
        message.push(output_message);
        const toolRequests = output_message.content;
        const toolMessage = toolRequests[0].text;
        console.log(toolMessage.replace(/<[^>]+>/g, ""));
        for (const toolRequest of toolRequests) {
          if (Object.hasOwn(toolRequest, "toolUse")) {
            const toolUse = toolRequest.toolUse;
            const sign = toolUse.input.sign;

```

```

    const toolUseID = toolUse.toolUseId;
    console.log(
      `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
    );
    if (toolUse.name === "top_song") {
      const toolResult = [];
      try {
        const top_song = await get_top_song(toolUse.input.sign).then(
          (top_song) => top_song,
        );
        const toolResult = {
          toolResult: {
            toolUseId: toolUseID,
            content: [
              {
                json: { song: top_song.song, artist: top_song.artist },
              },
            ],
          },
        };
        toolResultFinal.push(toolResult);
      } catch (err) {
        const toolResult = {
          toolUseId: toolUseID,
          content: [{ json: { text: err.message } }],
          status: "error",
        };
      }
    }
  }
}

const toolResultMessage = {
  role: "user",
  content: toolResultFinal,
};

// Step 4. Add the tool response to the conversation, and send it back to
Amazon Bedrock.

message.push(toolResultMessage);
await SendConversationtoBedrock(
  modelId,
  message,
  system_prompt,
  tool_config,

```

```

    );
  } catch (caught) {
    console.error(`${caught.message}`);
    throw caught;
  }
}

// 4. Publish the response.
if (response.stopReason === "end_turn") {
  const finalMessage = response.output.message.content[0].text;
  const messageToPrint = finalMessage.replace(/<[^>]+>/g);
  console.log(messageToPrint.replace(/<[^>]+>/g));
  return messageToPrint;
}
} catch (caught) {
  if (caught.name === "ModelNotReady") {
    console.log(
      `${caught.name} - Model not ready, please wait and try again.`
    );
    throw caught;
  }
  if (caught.name === "BedrockRuntimeException") {
    console.log(
      `${caught.name} - Error occurred while sending Converse request`
    );
    throw caught;
  }
}
}
}
await SendConversationtoBedrock(modelId, message, system_prompt, tool_config);

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Converse](#)를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Amazon Nova에 텍스트 메시지를 전송하고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Amazon Nova로 텍스트 메시지를 전송하고 응답 스트림을 실시간으로 처리합니다.

```
// This example demonstrates how to use the Amazon Nova foundation models
// to generate streaming text responses.
// It shows how to:
// - Set up the Amazon Bedrock runtime client
// - Create a message
// - Configure a streaming request
// - Process the streaming response

import {
  BedrockRuntimeClient,
  ConversationRole,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Step 1: Create the Amazon Bedrock runtime client
// Credentials will be automatically loaded from the environment
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Step 2: Specify which model to use
// Available Amazon Nova models and their characteristics:
// - Amazon Nova Micro: Text-only model optimized for lowest latency and cost
// - Amazon Nova Lite: Fast, low-cost multimodal model for image, video, and text
// - Amazon Nova Pro: Advanced multimodal model balancing accuracy, speed, and
//   cost
//
// For the most current model IDs, see:
// https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
const modelId = "amazon.nova-lite-v1:0";

// Step 3: Create the message
// The message includes the text prompt and specifies that it comes from the user
```

```
const inputText =
  "Describe the purpose of a 'hello world' program in one paragraph";
const message = {
  content: [{ text: inputText }],
  role: ConversationRole.USER,
};

// Step 4: Configure the streaming request
// Optional parameters to control the model's response:
// - maxTokens: maximum number of tokens to generate
// - temperature: randomness (max: 1.0, default: 0.7)
// OR
// - topP: diversity of word choice (max: 1.0, default: 0.9)
// Note: Use either temperature OR topP, but not both
const request = {
  modelId,
  messages: [message],
  inferenceConfig: {
    maxTokens: 500, // The maximum response length
    temperature: 0.5, // Using temperature for randomness control
    //topP: 0.9,      // Alternative: use topP instead of temperature
  },
};

// Step 5: Send and process the streaming request
// - Send the request to the model
// - Process each chunk of the streaming response
try {
  const response = await client.send(new ConverseStreamCommand(request));

  for await (const chunk of response.stream) {
    if (chunk.contentBlockDelta) {
      // Print each text chunk as it arrives
      process.stdout.write(chunk.contentBlockDelta.delta?.text || "");
    }
  }
} catch (error) {
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  process.exitCode = 1;
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ConverseStream](#)을 참조하세요.

시나리오: Converse API에서 도구 사용

다음 코드 예제에서는 애플리케이션, 생성형 AI 모델, 연결된 도구 또는 API 간에 일반적인 상호 작용을 구축하여 AI와 외부 환경 간의 상호 작용을 매개하는 방법을 보여줍니다. 외부 날씨 API를 AI 모델에 연결하는 예제를 사용하면 사용자 입력에 따라 실시간 날씨 정보를 제공할 수 있습니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배우보세요.

시나리오 흐름의 기본 실행입니다. 이 시나리오는 사용자, Amazon Bedrock Converse API 및 날씨 도구 간의 대화를 오케스트레이션합니다.

```
/* Before running this JavaScript code example, set up your development environment,
including your credentials.
This demo illustrates a tool use scenario using Amazon Bedrock's Converse API and a
weather tool.
The script interacts with a foundation model on Amazon Bedrock to provide weather
information based on user
input. It uses the Open-Meteo API (https://open-meteo.com) to retrieve current
weather data for a given location.*/

import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";
import data from "./questions.json" with { type: "json" };
import toolConfig from "./tool_config.json" with { type: "json" };
```

```

const __filename = fileURLToPath(import.meta.url);

const systemPrompt = [
  {
    text:
      "You are a weather assistant that provides current weather data for user-  

specified locations using only\n" +
      "the Weather_Tool, which expects latitude and longitude. Infer the coordinates  

from the location yourself.\n" +
      "If the user provides coordinates, infer the approximate location and refer to  

it in your response.\n" +
      "To use the tool, you strictly apply the provided tool specification.\n" +
      "If the user specifies a state, country, or region, infer the locations of  

cities within that state.\n" +
      "\n" +
      "- Explain your step-by-step process, and give brief updates before each step.  

\n" +
      "- Only use the Weather_Tool for data. Never guess or make up information. \n"
    +
      "- Repeat the tool use for subsequent requests if necessary.\n" +
      "- If the tool errors, apologize, explain weather is unavailable, and suggest  

other options.\n" +
      "- Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports  

concise. Sparingly use\n" +
      " emojis where appropriate.\n" +
      "- Only respond to weather queries. Remind off-topic users of your purpose.  

\n" +
      "- Never claim to search online, access external data, or use tools besides  

Weather_Tool.\n" +
      "- Complete the entire process until you have all required data before sending  

the complete response.",
  },
];

const tools_config = toolConfig;

/// Starts the conversation with the user and handles the interaction with Bedrock.
async function askQuestion(userMessage) {
  // The maximum number of recursive calls allowed in the tool use function.
  // This helps prevent infinite loops and potential performance issues.
  const max_recursions = 5;
  const messages = [
    {
      role: "user",

```



```

        content: [{ text: userMessage }],
    },
];
try {
    const response = await SendConversationtoBedrock(messages);
    await ProcessModelResponseAsync(response, messages, max_recurSIONs);
} catch (error) {
    console.log("error ", error);
}
}

// Sends the conversation, the system prompt, and the tool spec to Amazon Bedrock,
// and returns the response.
// param "messages" - The conversation history including the next message to send.
// return - The response from Amazon Bedrock.
async function SendConversationtoBedrock(messages) {
    const bedRockRuntimeClient = new BedrockRuntimeClient({
        region: "us-east-1",
    });
    try {
        const modelId = "amazon.nova-lite-v1:0";
        const response = await bedRockRuntimeClient.send(
            new ConverseCommand({
                modelId: modelId,
                messages: messages,
                system: systemPrompt,
                toolConfig: tools_config,
            }),
        );
        return response;
    } catch (caught) {
        if (caught.name === "ModelNotReady") {
            console.log(
                ``${caught.name}` - Model not ready, please wait and try again.",
            );
            throw caught;
        }
        if (caught.name === "BedrockRuntimeException") {
            console.log(
                ``${caught.name}` - "Error occurred while sending Converse request.",
            );
            throw caught;
        }
    }
}

```

```

}

// Processes the response received via Amazon Bedrock and performs the necessary
// actions based on the stop reason.
// param "response" - The model's response returned via Amazon Bedrock.
// param "messages" - The conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function ProcessModelResponseAsync(response, messages, max_recursions) {
  if (max_recursions <= 0) {
    await HandleToolUseAsync(response, messages);
  }
  if (response.stopReason === "tool_use") {
    await HandleToolUseAsync(response, messages, max_recursions - 1);
  }
  if (response.stopReason === "end_turn") {
    const messageToPrint = response.output.message.content[0].text;
    console.log(messageToPrint.replace(/<[^>]+>/g, ""));
  }
}

// Handles the tool use case by invoking the specified tool and sending the tool's
// response back to Bedrock.
// The tool response is appended to the conversation, and the conversation is sent
// back to Amazon Bedrock for further processing.
// param "response" - the model's response containing the tool use request.
// param "messages" - the conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function HandleToolUseAsync(response, messages, max_recursions) {
  const toolResultFinal = [];
  try {
    const output_message = response.output.message;
    messages.push(output_message);
    const toolRequests = output_message.content;
    const toolMessage = toolRequests[0].text;
    console.log(toolMessage.replace(/<[^>]+>/g, ""));
    for (const toolRequest of toolRequests) {
      if (Object.hasOwn(toolRequest, "toolUse")) {
        const toolUse = toolRequest.toolUse;
        const latitude = toolUse.input.latitude;
        const longitude = toolUse.input.longitude;
        const toolUseID = toolUse.toolUseId;
        console.log(
          `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
        );
        if (toolUse.name === "Weather_Tool") {

```

```

    try {
      const current_weather = await callWeatherTool(
        longitude,
        latitude,
      ).then((current_weather) => current_weather);
      const currentWeather = current_weather;
      const toolResult = {
        toolResult: {
          toolUseId: toolUseID,
          content: [{ json: currentWeather }],
        },
      };
      toolResultFinal.push(toolResult);
    } catch (err) {
      console.log("An error occurred. ", err);
    }
  }
}

const toolResultMessage = {
  role: "user",
  content: toolResultFinal,
};
messages.push(toolResultMessage);
// Send the conversation to Amazon Bedrock
await ProcessModelResponseAsync(
  await SendConversationtoBedrock(messages),
  messages,
);
} catch (error) {
  console.log("An error occurred. ", error);
}
}
// Call the Weathertool.
// param = longitude of location
// param = latitude of location
async function callWeatherTool(longitude, latitude) {
  // Open-Meteo API endpoint
  const apiUrl = `https://api.open-meteo.com/v1/forecast?latitude=${latitude}&longitude=${longitude}&current_weather=true`;

  // Fetch the weather data.
  return fetch(apiUrl)

```

```

        .then((response) => {
            return response.json().then((current_weather) => {
                return current_weather;
            });
        })
        .catch((error) => {
            console.error("Error fetching weather data:", error);
        });
    }
    /**
     * Used repeatedly to have the user press enter.
     * @type {ScenarioInput}
     */
    const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
        type: "input",
        default: "",
    });

    const greet = new ScenarioOutput(
        "greet",
        "Welcome to the Amazon Bedrock Tool Use demo! \n" +
        "This assistant provides current weather information for user-specified locations. " +
        "You can ask for weather details by providing the location name or coordinates." +
        "Weather information will be provided using a custom Tool and open-meteo API." +
        "For the purposes of this example, we'll use in order the questions in ./questions.json :\n" +
        "What's the weather like in Seattle? " +
        "What's the best kind of cat? " +
        "Where is the warmest city in Washington State right now? " +
        "What's the warmest city in California right now?\n" +
        "To exit the program, simply type 'x' and press Enter.\n" +
        "Have fun and experiment with the app by editing the questions in ./questions.json! " +
        "P.S.: You're not limited to single locations, or even to using English! ",

        { header: true },
    );

    const displayAskQuestion1 = new ScenarioOutput(
        "displayAskQuestion1",
        "Press enter to ask question number 1 (default is 'What's the weather like in Seattle?')",
    );

```

```
const askQuestion1 = new ScenarioAction(
  "askQuestion1",
  async (** @type {State} */ state) => {
    const userMessage1 = data.questions["question-1"];
    await askQuestion(userMessage1);
  },
);

const displayAskQuestion2 = new ScenarioOutput(
  "displayAskQuestion2",
  "Press enter to ask question number 2 (default is 'What's the best kind of cat?')",
);

const askQuestion2 = new ScenarioAction(
  "askQuestion2",
  async (** @type {State} */ state) => {
    const userMessage2 = data.questions["question-2"];
    await askQuestion(userMessage2);
  },
);

const displayAskQuestion3 = new ScenarioOutput(
  "displayAskQuestion3",
  "Press enter to ask question number 3 (default is 'Where is the warmest city in Washington State right now?')",
);

const askQuestion3 = new ScenarioAction(
  "askQuestion3",
  async (** @type {State} */ state) => {
    const userMessage3 = data.questions["question-3"];
    await askQuestion(userMessage3);
  },
);

const displayAskQuestion4 = new ScenarioOutput(
  "displayAskQuestion4",
  "Press enter to ask question number 4 (default is 'What's the warmest city in California right now?')",
);

const askQuestion4 = new ScenarioAction(
  "askQuestion4",
```

```

    async (/** @type {State} */ state) => {
      const userMessage4 = data.questions["question-4"];
      await askQuestion(userMessage4);
    },
  );

  const goodbye = new ScenarioOutput(
    "goodbye",
    "Thank you for checking out the Amazon Bedrock Tool Use demo. We hope you\n" +
      "learned something new, or got some inspiration for your own apps today!\n" +
      "For more Bedrock examples in different programming languages, have a look at:\n" +
      "\n" +
      "https://docs.aws.amazon.com/bedrock/latest/userguide/service_code_examples.html",
  );

  const myScenario = new Scenario("Converse Tool Scenario", [
    greet,
    pressEnter,
    displayAskQuestion1,
    askQuestion1,
    pressEnter,
    displayAskQuestion2,
    askQuestion2,
    pressEnter,
    displayAskQuestion3,
    askQuestion3,
    pressEnter,
    displayAskQuestion4,
    askQuestion4,
    pressEnter,
    goodbye,
  ]);

  /** @type {{ stepHandlerOptions: StepHandlerOptions }} */
  export const main = async (stepHandlerOptions) => {
    await myScenario.run(stepHandlerOptions);
  };

  // Invoke main function if this file was run directly.
  if (process.argv[1] === fileURLToPath(import.meta.url)) {
    const { values } = parseArgs({
      options: {
        yes: {

```

```

        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Converse](#)를 참조하세요.

Amazon Nova Canvas

InvokeModel

다음 코드 예제에서는 이미지 생성을 위해 Amazon Bedrock에서 Amazon Nova Canvas를 간접적으로 호출하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon Nova Canvas로 이미지를 생성합니다.

```

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
import { saveImage } from "../utils/image-creation.js";
import { fileURLToPath } from "node:url";

/**
 * This example demonstrates how to use Amazon Nova Canvas to generate images.
 * It shows how to:
 * - Set up the Amazon Bedrock runtime client
 * - Configure the image generation parameters
 * - Send a request to generate an image

```

```
* - Process the response and handle the generated image
*
* @returns {Promise<string>} Base64-encoded image data
*/
export const invokeModel = async () => {
  // Step 1: Create the Amazon Bedrock runtime client
  // Credentials will be automatically loaded from the environment
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Step 2: Specify which model to use
  // For the latest available models, see:
  // https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
  const modelId = "amazon.nova-canvas-v1:0";

  // Step 3: Configure the request payload
  // First, set the main parameters:
  // - prompt: Text description of the image to generate
  // - seed: Random number for reproducible generation (0 to 858,993,459)
  const prompt = "A stylized picture of a cute old steampunk robot";
  const seed = Math.floor(Math.random() * 858993460);

  // Then, create the payload using the following structure:
  // - taskType: TEXT_IMAGE (specifies text-to-image generation)
  // - textToImageParams: Contains the text prompt
  // - imageGenerationConfig: Contains optional generation settings (seed, quality,
  // etc.)
  // For a list of available request parameters, see:
  // https://docs.aws.amazon.com/nova/latest/userguide/image-gen-req-resp-
  // structure.html
  const payload = {
    taskType: "TEXT_IMAGE",
    textToImageParams: {
      text: prompt,
    },
    imageGenerationConfig: {
      seed,
      quality: "standard",
    },
  };

  // Step 4: Send and process the request
  // - Embed the payload in a request object
  // - Send the request to the model
  // - Extract and return the generated image data from the response
```



```

try {
  const request = {
    modelId,
    body: JSON.stringify(payload),
  };
  const response = await client.send(new InvokeModelCommand(request));

  const decodedResponseBody = new TextDecoder().decode(response.body);
  // The response includes an array of base64-encoded PNG images
  /** @type {{images: string[]}} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.images[0]; // Base64-encoded image data
} catch (error) {
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  throw error;
}
};

// If run directly, execute the example and save the generated image
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("Generating image. This may take a few seconds...");
  invokeModel()
    .then(async (imageData) => {
      const imagePath = await saveImage(imageData, "nova-canvas");
      // Example path: javascriptv3/example_code/bedrock-runtime/output/nova-canvas/
      // image-01.png
      console.log(`Image saved to: ${imagePath}`);
    })
    .catch((error) => {
      console.error("Execution failed:", error);
      process.exitCode = 1;
    });
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [InvokeModel](#)을 참조하세요.

Anthropic Claude

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보냅니다.

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
```

```

    modelId,
    messages: conversation,
    inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
  });

  try {
    // Send the command to the model and wait for the response
    const response = await client.send(command);

    // Extract and print the response text.
    const responseText = response.output.message.content[0].text;
    console.log(responseText);
  } catch (err) {
    console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
    process.exit(1);
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Converse](#)를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```

// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,

```

```
    ConverseStreamCommand,
  } from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ConverseStream](#)을 참조하세요.

InvokeModel

다음 코드 예제에서는 Invoke Model API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Invoke Model API를 사용하여 텍스트 메시지를 보냅니다.

```
import { fileURLToPath } from "node:url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */
```

```
/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
"anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
}
```

```
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 * "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
```

```

/** @type Chunk */
const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
const chunk_type = chunk.type;

if (chunk_type === "content_block_delta") {
  const text = chunk.delta.text;
  completeMessage = completeMessage + text;
  process.stdout.write(text);
}
}

// Return the final response
return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(`\n${"-".repeat(53)}`);
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [InvokeModel](#)을 참조하세요.

InvokeModelWithResponseStream

다음 코드 예제에서는 Invoke Model API를 사용하여 Anthropic Claude 모델에 텍스트 메시지를 보내고 응답 스트림을 프린트하는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Invoke Model API를 사용하여 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
import { fileURLToPath } from "node:url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
```

```

* https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
*
* @param {string} prompt - The input text prompt for the model to complete.
* @param {string} [modelId] - The ID of the model to use. Defaults to
"anthropic.claude-3-haiku-20240307-v1:0".
*/
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 */

```

```

* To learn more about the Anthropic Messages API, go to:
* https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
*
* @param {string} prompt - The input text prompt for the model to complete.
* @param {string} [modelId] - The ID of the model to use. Defaults to
"anthropic.claude-3-haiku-20240307-v1:0".
*/
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;

    if (chunk_type === "content_block_delta") {

```

```

        const text = chunk.delta.text;
        completeMessage = completeMessage + text;
        process.stdout.write(text);
    }
}

// Return the final response
return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
    const prompt = 'Write a paragraph starting with: "Once upon a time...";
    const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
    console.log(`Prompt: ${prompt}`);
    console.log(`Model ID: ${modelId}`);

    try {
        console.log("-".repeat(53));
        const response = await invokeModel(prompt, modelId);
        console.log(`\n${"-".repeat(53)}`);
        console.log("Final structured response:");
        console.log(response);
    } catch (err) {
        console.log(`\n${err}`);
    }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [InvokeModelWithResponseStream](#)을 참조하세요.

Cohere Command

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Cohere Command에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Cohere Command로 텍스트 메시지를 보냅니다.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1.0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
}
```

```
// Extract and print the response text.
const responseText = response.output.message.content[0].text;
console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Converse](#)를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Cohere Command에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Cohere Command에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";
```

```
// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ConverseStream](#)을 참조하세요.

Meta Llama

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Meta Llama에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Meta Llama에 텍스트 메시지를 보냅니다.

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
}
```



```
// Extract and print the response text.
const responseText = response.output.message.content[0].text;
console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Converse](#)를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Meta Llama에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Meta Llama에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";
```

```
// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ConverseStream](#)을 참조하세요.

InvokeModel

다음 코드 예제에서는 Invoke Model API를 사용하여 Meta Llama에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Invoke Model API를 사용하여 텍스트 메시지를 보냅니다.

```
// Send a prompt to Meta Llama 3 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 70B Instruct.
const modelId = "meta.llama3-70b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|><|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};
```

```
// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [InvokeModel](#)을 참조하세요.

InvokeModelWithResponseStream

다음 코드 예제에서는 Invoke Model API를 사용하여 Meta Llama에 텍스트 메시지를 보내고 응답 스트림을 프린트하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Invoke Model API를 사용하여 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
// Send a prompt to Meta Llama 3 and print the response stream in real-time.
```

```
import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 70B Instruct.
const modelId = "meta.llama3-70b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|><|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Extract and print the response stream in real-time.
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
```

```
const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
if (chunk.generation) {
  process.stdout.write(chunk.generation);
}
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [InvokeModelWithResponseStream](#)을 참조하세요.

Mistral AI

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Mistral에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Mistral에 텍스트 메시지를 보냅니다.

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });
```

```
// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Converse](#)를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Mistral에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Mistral에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
```



```
// Send the command to the model and wait for the response
const response = await client.send(command);

// Extract and print the streamed response text in real-time.
for await (const item of response.stream) {
  if (item.contentBlockDelta) {
    process.stdout.write(item.contentBlockDelta.delta?.text);
  }
}
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ConverseStream](#)을 참조하세요.

InvokeModel

다음 코드 예제에서는 Invoke Model API를 사용하여 Mistral 모델에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Invoke Model API를 사용하여 텍스트 메시지를 보냅니다.

```
import { fileURLToPath } from "node:url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
```

```
/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes a Mistral 7B Instruct model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 * "mistral.mistral-7b-instruct-v0:2".
 */
export const invokeModel = async (
  prompt,
  modelId = "mistral.mistral-7b-instruct-v0:2",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `<s>[INST] ${prompt} [/INST]`;

  // Prepare the payload.
  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
```

```

const responseBody = JSON.parse(decodedResponseBody);
return responseBody.outputs[0].text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.MISTRAL_7B.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [InvokeModel](#)을 참조하세요.

SDK for JavaScript(v3)를 사용한 Amazon Bedrock Agents 예제

다음 코드 예제에서는 Amazon Bedrock Agents에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)

시작하기

Amazon Bedrock Agents 시작

다음 코드 예제에서는 Amazon Bedrock Agents 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node: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](#)

작업

CreateAgent

다음 코드 예시는 CreateAgent의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

에이전트를 생성합니다.

```
import { fileURLToPath } from "node: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 참조의 [CreateAgent](#)를 참조하세요.

DeleteAgent

다음 코드 예시는 DeleteAgent의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

에이전트를 삭제합니다.

```
import { fileURLToPath } from "node: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 참조의 [DeleteAgent](#)를 참조하세요.

GetAgent

다음 코드 예시는 GetAgent의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

에이전트를 가져옵니다.

```
import { fileURLToPath } from "node: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 참조의 [GetAgent](#)를 참조하세요.

ListAgentActionGroups

다음 코드 예시는 ListAgentActionGroups의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

에이전트의 작업 그룹을 나열합니다.

```
import { fileURLToPath } from "node: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 참조의 [ListAgentActionGroups](#)를 참조하세요.

ListAgents

다음 코드 예시는 ListAgents의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

계정에 속한 에이전트를 나열합니다.

```
import { fileURLToPath } from "node: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 참조의 [ListAgents](#)를 참조하세요.

SDK for JavaScript(v3)를 사용한 Amazon Bedrock Agents Runtime 예제

다음 코드 예제에서는 Amazon Bedrock Agents 런타임과 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

InvokeAgent

다음 코드 예시는 InvokeAgent의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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 (const 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 "node: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 참조의 [InvokeAgent](#)를 참조하세요.

InvokeFlow

다음 코드 예시는 InvokeFlow의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { fileURLToPath } from "node:url";

import {
  BedrockAgentRuntimeClient,
  InvokeFlowCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

```

```

/**
 * Invokes an alias of a flow to run the inputs that you specify and return
 * the output of each node as a stream.
 *
 * @param {{
 *   flowIdentifier: string,
 *   flowAliasIdentifier: string,
 *   prompt?: string,
 *   region?: string
 * }} options
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").FlowNodeOutput>} An
 * object containing information about the output from flow invocation.
 */
export const invokeBedrockFlow = async ({
  flowIdentifier,
  flowAliasIdentifier,
  prompt = "Hi, how are you?",
  region = "us-east-1",
}) => {
  const client = new BedrockAgentRuntimeClient({ region });

  const command = new InvokeFlowCommand({
    flowIdentifier,
    flowAliasIdentifier,
    inputs: [
      {
        content: {
          document: prompt,
        },
        nodeName: "FlowInputNode",
        nodeOutputName: "document",
      },
    ],
  });

  let flowResponse = {};
  const response = await client.send(command);

  for await (const chunkEvent of response.responseStream) {
    const { flowOutputEvent, flowCompletionEvent } = chunkEvent;

    if (flowOutputEvent) {
      flowResponse = { ...flowResponse, ...flowOutputEvent };
    }
  }
}

```

```
        console.log("Flow output event:", flowOutputEvent);
    } else if (flowCompletionEvent) {
        flowResponse = { ...flowResponse, ...flowCompletionEvent };
        console.log("Flow completion event:", flowCompletionEvent);
    }
}

return flowResponse;
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
    isMain,
    validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
    const options = {
        flowIdentifier: {
            type: "string",
            required: true,
        },
        flowAliasIdentifier: {
            type: "string",
            required: true,
        },
        prompt: {
            type: "string",
        },
        region: {
            type: "string",
        },
    };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        invokeBedrockFlow(results.values);
    } else {
```

```
    console.error(errors.join("\n"));
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조에서 [InvokeFlow](#)를 참조하세요.

SDK for JavaScript (v3)를 사용한 CloudWatch 예제

다음 코드 예제에서는 CloudWatch에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

DeleteAlarms

다음 코드 예시는 DeleteAlarms의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteAlarms](#)를 참조하세요.

DescribeAlarmsForMetric

다음 코드 예시는 DescribeAlarmsForMetric의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeAlarmsForMetric](#)을 참조하세요.

DisableAlarmActions

다음 코드 예시는 DisableAlarmActions의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DisableAlarmActions](#)을 참조하세요.

EnableAlarmActions

다음 코드 예시는 EnableAlarmActions의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [EnableAlarmActions](#)를 참조하세요.

ListMetrics

다음 코드 예시는 ListMetrics의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK 및 클라이언트 모듈을 가져오고 API를 호출합니다.

```
import {
  CloudWatchServiceException,
  ListMetricsCommand,
} from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = async () => {
  // 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",
  });

  try {
    const response = await client.send(command);
    console.log(`Metrics count: ${response.Metrics?.length}`);
    return response;
  } catch (caught) {
    if (caught instanceof CloudWatchServiceException) {
      console.error(`Error from CloudWatch. ${caught.name}: ${caught.message}`);
    } else {
```

```

        throw caught;
    }
}
};

```

별도의 모듈에서 클라이언트를 생성하고 내보냅니다.

```

import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListMetrics](#)를 참조하세요.

PutMetricAlarm

다음 코드 예시는 PutMetricAlarm의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutMetricAlarm](#)을 참조하세요.

PutMetricData

다음 코드 예시는 PutMetricData의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutMetricData](#)를 참조하세요.

SDK for JavaScript (v3)를 사용한 CloudWatch Events 예제

다음 코드 예제에서는 CloudWatch Events와 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

PutEvents

다음 코드 예시는 PutEvents의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutEvents](#)를 참조하세요.

PutRule

다음 코드 예시는 PutRule의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutRule](#)을 참조하세요.

PutTargets

다음 코드 예시는 PutTargets의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutTargets](#)를 참조하세요.

SDK for JavaScript (v3)를 사용한 CloudWatch Logs 예제

다음 코드 예제에서는 CloudWatch Logs와 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)
- [시나리오](#)

작업

CreateLogGroup

다음 코드 예시는 CreateLogGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateLogGroup](#) 참조하세요.

DeleteLogGroup

다음 코드 예시는 DeleteLogGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteLogGroup](#) 참조하세요.

DeleteSubscriptionFilter

다음 코드 예시는 DeleteSubscriptionFilter의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteSubscriptionFilter](#) 참조하세요.

DescribeLogGroups

다음 코드 예시는 DescribeLogGroups의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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?.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }

  console.log(logGroups);
  return logGroups;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeLogGroups](#) 참조하세요.

DescribeSubscriptionFilters

다음 코드 예시는 DescribeSubscriptionFilters의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeSubscriptionFilters](#) 참조하세요.

GetQueryResults

다음 코드 예시는 GetQueryResults의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetQueryResults](#)를 참조하세요.

PutSubscriptionFilter

다음 코드 예시는 PutSubscriptionFilter의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [PutSubscriptionFilter](#) 참조하세요.

StartLiveTail

다음 코드 예시는 StartLiveTail의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

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);

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [StartLiveTail](#)을 참조하세요.

StartQuery

다음 코드 예시는 StartQuery의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [StartQuery](#)를 참조하세요.

시나리오

대용량 쿼리 실행

다음 코드 예시에서는 CloudWatch Logs를 사용하여 1만 개 이상의 레코드를 쿼리하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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(Number.parseInt(process.env.QUERY_START_DATE)),
    new Date(Number.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]);
}

/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {

```



```

    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }

  /**
   * 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 [];
      }
      throw err;
    }
  }

  /**
   * 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;
    }
  }

  /**
   * 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](#)

예약된 이벤트를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon EventBridge 예약 이벤트에서 호출된 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS Lambda 함수를 호출하는 Amazon EventBridge 예약 이벤트를 생성하는 방법을 보여줍니다. Lambda 함수가 간접 호출될 때 cron 표현식을 사용하여 일정을 예약하도록 EventBridge를 구성합니다. 이 예제에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

SDK for JavaScript (v3)를 사용한 CodeBuild 예제

다음 코드 예제에서는 CodeBuild에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

CreateProject

다음 코드 예시는 CreateProject의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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-xxxxxx-xxxxxx',
//          name: 'MyCodeBuilder',
//          namespaceType: 'NONE',
//          packaging: 'NONE',
//          type: 'S3'
//      },
//      badge: { badgeEnabled: false },
//      cache: { type: 'NO_CACHE' },
//      created: 2023-08-18T14:46:48.979Z,
//      encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
//      environment: {
//          computeType: 'BUILD_GENERAL1_SMALL',
//          environmentVariables: [],
//          image: 'aws/codebuild/standard:7.0',
//          imagePullCredentialsType: 'CODEBUILD',
//          privilegedMode: false,
//          type: 'LINUX_CONTAINER'
//      },
//      lastModified: 2023-08-18T14:46:48.979Z,
//      name: 'MyCodeBuilder',
//      projectVisibility: 'PRIVATE',
//      queuedTimeoutInMinutes: 480,
//      serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
//      source: {
//          insecureSsl: false,
//          location: 'https://...',
//          reportBuildStatus: false,
//          type: 'GITHUB'
//      },
//      timeoutInMinutes: 60
//  }
//  }
return response;
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateProject](#)를 참조하세요.

SDK for JavaScript(v3)를 사용한 Amazon Cognito ID 예제

다음 코드 예제에서는 Amazon Cognito 자격 증명과 함께 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

Amazon Textract 탐색기 애플리케이션 생성

다음 코드 예제에서는 대화형 애플리케이션을 통해 Amazon Textract 출력을 탐색하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript 를 사용하여 Amazon Textract를 사용하여 문서 이미지에서 데이터를 추출하고 대화형 웹 페이지에 표시하는 React 애플리케이션을 빌드하는 방법을 보여줍니다. 이 예제는 웹 브라우저에서 실행되며 자격 증명을 위해 인증된 Amazon Cognito 자격 증명이 필요합니다. 이 애플리케이션은 스토리지로 Amazon Simple Storage Service(Amazon 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

SDK for JavaScript (v3)를 사용한 Amazon Cognito 자격 증명 공급자 예제

다음 코드 예제에서는 Amazon Cognito 자격 증명 공급자와 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)
- [시나리오](#)

시작하기

Hello Amazon Cognito

다음 코드 예시에서는 Amazon Cognito 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListFunctions](#)를 참조하세요.

작업

AdminGetUser

다음 코드 예시는 AdminGetUser의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [AdminGetUser](#)를 참조하세요.

AdminInitiateAuth

다음 코드 예시는 AdminInitiateAuth의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [AdminInitiateAuth](#)를 참조하세요.

AdminRespondToAuthChallenge

다음 코드 예시는 AdminRespondToAuthChallenge의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [AdminRespondToAuthChallenge](#)를 참조하세요.

AssociateSoftwareToken

다음 코드 예시는 AssociateSoftwareToken의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [AssociateSoftwareToken](#)을 참조하세요.

ConfirmDevice

다음 코드 예시는 ConfirmDevice의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ConfirmDevice](#)를 참조하세요.

ConfirmSignUp

다음 코드 예시는 ConfirmSignUp의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ConfirmSignUp](#)을 참조하세요.

DeleteUser

다음 코드 예시는 DeleteUser의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
 * provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteUser](#)를 참조하세요.

InitiateAuth

다음 코드 예시는 InitiateAuth의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [InitiateAuth](#)를 참조하세요.

ListUsers

다음 코드 예시는 ListUsers의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListUsers](#)를 참조하세요.

ResendConfirmationCode

다음 코드 예시는 ResendConfirmationCode의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ResendConfirmationCode](#)를 참조하세요.

RespondToAuthChallenge

다음 코드 예시는 RespondToAuthChallenge의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [RespondToAuthChallenge](#)를 참조하세요.

SignUp

다음 코드 예시는 SignUp의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [SignUp](#)를 참조하세요.

UpdateUserPool

다음 코드 예시는 UpdateUserPool의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
```

```

        UserPoolId: userPoolId,
        LambdaConfig: {
            PreSignUp: handlerArn,
        },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
} catch (err) {
    return [null, err];
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateUserPool](#)을 참조하세요.

VerifySoftwareToken

다음 코드 예시는 VerifySoftwareToken의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [VerifySoftwareToken](#)을 참조하세요.

시나리오

Lambda 함수를 사용하여 알려진 사용자를 자동으로 확인

다음 코드 예제는 Lambda 함수를 사용하여 알려진 Amazon Cognito 사용자를 자동으로 확인하는 방법을 보여줍니다.

- PreSignUp 트리거에 대해 Lambda 함수를 호출하도록 사용자 풀을 구성합니다.
- Amazon Cognito를 사용하여 사용자 가입시키기
- Lambda 함수는 DynamoDB 테이블을 스캔하고 알려진 사용자를 자동으로 확인합니다.
- 새 사용자로 로그인한 다음 리소스를 정리합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

대화형 '시나리오' 실행을 구성합니다. JavaScript(v3) 예제에서는 시나리오 실행기를 공유하여 복잡한 예제를 간소화합니다. 전체 소스 코드는 GitHub에 있습니다.

```

import { AutoConfirm } from "../scenario-auto-confirm.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {

```

```

    errors: [],
    users: [
      {
        UserName: "test_user_1",
        UserEmail: "test_email_1@example.com",
      },
      {
        UserName: "test_user_2",
        UserEmail: "test_email_2@example.com",
      },
      {
        UserName: "test_user_3",
        UserEmail: "test_email_3@example.com",
      },
    ],
  };

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito authentication behavior.",
  });
}

```

이 시나리오에서는 알려진 사용자를 자동으로 확인하는 방법을 보여줍니다. 여기에서는 예제 단계를 오케스트레이션합니다.

```

import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
} from "./steps-common.js";
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { Username: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,

```

```

*   AutoConfirmHandlerName?: string
* }} State
*/

const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.` ,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",

```

```

    "Setting up the PreSignUp trigger for the Cognito User Pool.",
    { skipWhen: skipWhenErrors },
  );

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (/** @type {State} */ state) => {
    const [_, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
"${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (/** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (/** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",

```



```

async (/** @type {State} */ state) => {
  const [user, err] = await getUser({
    region: state.stackRegion,
    userPoolId: state.UserPoolId,
    username: state.selectedUser,
  });

  if (err?.name === "UserNotFoundException") {
    // Do nothing. We're not expecting the user to exist before
    // sign up is complete.
    return;
  }

  if (err) {
    state.errors.push(err);
    return;
  }

  if (user) {
    state.errors.push(
      new Error(
        `The user "${state.selectedUser}" already exists in the user pool
        "${state.UserPoolId}".`,
      ),
    );
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
  numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

```

```
const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({
        region: state.stackRegion,
        userPoolClientId: state.UserPoolClientId,
        username: state.selectedUser,
        email: state.users.find((u) => u.UserName === state.selectedUser)
          .UserEmail,
        password,
      });

    let [_, err] = await signUp(state.password);

    while (err?.name === "InvalidPasswordException") {
      console.warn("The password you entered was invalid.");
      await createPassword.handle(state);
      [_, err] = await signUp(state.password);
    }

    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (** @type {State} */ state) =>
    `${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);
  },
);
```

```

const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
  functionName: state.AutoConfirmHandlerName,
  region: state.stackRegion,
});
if (logStreamErr) {
  state.errors.push(logStreamErr);
  return;
}

console.log(
  `Getting some recent events from log stream "${logStream.logStreamName}"`,
);
const [logEvents, logEventsErr] = await getLogEvents({
  functionName: state.AutoConfirmHandlerName,
  region: state.stackRegion,
  eventCount: 10,
  logStreamName: logStream.logStreamName,
});
if (logEventsErr) {
  state.errors.push(logEventsErr);
  return;
}

console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
},
{ skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });
  });

```

```
    if (err?.name === "PasswordResetRequiredException") {
      state.errors.push(new Error("Please reset your password."));
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with: ${state.token.slice(0, 11)}`,
  { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
```

```

    },
  );

const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State} */ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
    ],
  );

```

```

    logErrors,
  ],
  context,
);

```

다음은 다른 시나리오와 공유되는 단계입니다.

```

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",

```

```

    "Enter the region of the stack you deployed earlier.",
    { type: "input", default: "us-east-1" },
  );

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);

```

Lambda 함수에서 PreSignUp 트리거에 대한 핸들러.

```

import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "../user-repository";
import { DynamoDBUserRepository } from "../user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user '${event.userName}'`,
    );

    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }
  }
}

```

```
const eventEmail = this.getEventUserEmail(event);
console.log(`Looking up email ${eventEmail}.`);
const storedUserInfo =
  await this.userRepository.getUserInfoByEmail(eventEmail);

if (!storedUserInfo) {
  console.log(
    `Email ${eventEmail} not found. Email verification is required.`,
  );
  return event;
}

if (storedUserInfo.UserName !== event.userName) {
  console.log(
    `UserEmail ${eventEmail} found, but stored UserName
'${storedUserInfo.UserName}' does not match supplied UserName '${event.userName}'.
Verification is required.`,
  );
} else {
  console.log(
    `UserEmail ${eventEmail} found with matching UserName
${storedUserInfo.UserName}. User is confirmed.`,
  );
  event.response.autoConfirmUser = true;
  event.response.autoVerifyEmail = true;
}
return event;
}
}

const createPreSignUpHandler = (): PreSignUpHandler => {
  const tableName = process.env.TABLE_NAME;
  if (!tableName) {
    throw new Error("TABLE_NAME environment variable is not set");
  }

  const userRepository = new DynamoDBUserRepository(tableName);
  return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```


CloudWatch Logs 작업의 모듈.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
  unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
```

```

*   functionName: string,
*   logStreamName: string,
*   eventCount: number,
*   region: string
* }} config
* @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[] |
null, unknown]>}
*/
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};

```

Amazon Cognito 작업 모듈.

```

import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

```

```

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 *   password: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({

```

```

    region,
    userPoolClientId,
    username,
    email,
    password,
  }) => {
    try {
      const cognitoClient = new CognitoIdentityProviderClient({
        region,
      });

      const response = await cognitoClient.send(
        new SignUpCommand({
          ClientId: userPoolClientId,
          Username: username,
          Password: password,
          UserAttributes: [{ Name: "email", Value: email }],
        }),
      );
      return [response, null];
    } catch (err) {
      return [null, err];
    }
  };
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 * flow.
 * @param {{ region: string, clientId: string, username: string, password: string }}
 * config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
 * provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  }
};

```

```

    } catch (err) {
      return [null, err];
    }
  };

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

```

DynamoDB 작업의 모듈.

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

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }}
  config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
  null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [DeleteUser](#)

- [InitiateAuth](#)
- [SignUp](#)
- [UpdateUserPool](#)

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.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

최상의 경험을 위해 GitHub 리포지토리를 복제하고 이 예제를 실행하세요. 다음 코드는 전체 예제 애플리케이션의 샘플을 나타냅니다.

```
import { logger } 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);
    logger.log("Signing up.");
    await signUp({ clientId, username, password, email });
    logger.log(`Signed up. A confirmation email has been sent to: ${email}.`);
    logger.log(
      `Run 'confirm-sign-up ${username} <code>' to confirm your account.`,
    );
  } catch (err) {
    logger.error(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 { logger } 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);
```

```

    logger.log("Confirming user.");
    await confirmSignUp({ clientId, username, code });
    logger.log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
in.` ,
    );
  } catch (err) {
    logger.error(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 { logger } 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);
  }
};
```

```

    logger.log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      logger.log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      logger.log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    logger.error(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 { logger } 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);

    logger.log("Successfully authenticated.");
  } catch (err) {
    logger.error(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 { logger } 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);  
    }  
  };  
}
```

```
    logger.log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    logger.log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    logger.error(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 항목을 참조하세요.
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)

- [ListUsers](#)
- [ResendConfirmationCode](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

SDK for JavaScript(v3)를 사용한 Amazon Comprehend 예제

다음 코드 예제에서는 Amazon Comprehend에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

Amazon Transcribe 스트리밍 앱 구축

다음 코드 예제에서는 라이브 오디오를 실시간으로 기록, 변환 및 번역하고 결과를 이메일로 보내는 앱을 구축하는 방법을 보여줍니다.

SDK for JavaScript (v3)

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 [GitHub](#)에서 전체 예시를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Comprehend

- Amazon SES
- Amazon Transcribe
- Amazon Translate

Amazon Lex 챗봇 구축

다음 코드 예제에서는 챗봇을 만들어 웹사이트 방문자를 참여시키는 방법을 보여줍니다.

SDK for JavaScript(v3)

Amazon Lex API를 사용하여 웹 애플리케이션 내에 챗봇을 구축하여 웹 사이트 방문자의 참여를 유도하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 AWS SDK for JavaScript 개발자 안내서의 [Amazon Lex 챗봇 구축](#) 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

고객 피드백 분석을 위한 애플리케이션 생성

다음 코드 예제에서는 고객 의견 카드를 분석하고, 원어에서 번역하고, 감정을 파악하고, 번역된 텍스트에서 오디오 파일을 생성하는 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 코드와 배포 지침은 [GitHub](#)의 프로젝트를 참조하세요. 다음 발췌문은 Lambda 함수 내에서 AWS SDK for JavaScript가 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 예제에서 사용되는 서비스

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

SDK for JavaScript(v3)를 사용한 Amazon DocumentDB 예제

다음 코드 예제에서는 Amazon DocumentDB에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [서버리스 예제](#)

서버리스 예제

Amazon DocumentDB 트리거에서 간접적으로 Lambda 함수 호출

다음 코드 예제에서는 DocumentDB 변경 스트림에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DocumentDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 Amazon DocumentDB 이벤트 소비

```
console.log('Loading function');
exports.handler = async (event, context) => {
  event.events.forEach(record => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record) => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
    2));
};
```

TypeScript를 사용하여 Lambda로 Amazon DocumentDB 이벤트 소비

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';
```

```
console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

SDK for JavaScript (v3)를 사용한 DynamoDB 예

다음 코드 예제에서는 DynamoDB에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)

- [시나리오](#)
- [서버리스 예제](#)

시작하기

Hello DynamoDB

다음 코드 예제는 DynamoDB를 사용하여 시작하는 방법을 보여 줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

에서 DynamoDB 작업에 대한 자세한 내용은 [JavaScript를 사용한 DynamoDB 프로그래밍](#)을 AWS SDK for JavaScript참조하세요.

```
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 참조의 [ListTables](#)를 참조하세요.

기본 사항

기본 사항 알아보기

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.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { readFileSync } from "node: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 B00L) 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 docClient.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](#)

작업

BatchExecuteStatement

다음 코드 예시는 BatchExecuteStatement의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [BatchExecuteStatement](#)를 참조하세요.

BatchGetItem

다음 코드 예시는 BatchGetItem의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 [BatchGet](#)을 참조하세요.

```
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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [BatchGetItem](#)을 참조하세요.

BatchWriteItem

다음 코드 예시는 BatchWriteItem의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 [BatchWrite](#)를 참조하세요.

```

import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node: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 참조의 [BatchWriteItem](#)을 참조하세요.

CreateTable

다음 코드 예시는 CreateTable의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

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

```
console.log(response);
return response;
};
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateTable](#)을 참조하세요.

DeleteItem

다음 코드 예시는 DeleteItem의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 [DeleteCommand](#)를 참조하세요.

```
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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteItem](#)을 참조하세요.

DeleteTable

다음 코드 예시는 DeleteTable의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteTable](#)을 참조하세요.

DescribeTable

다음 코드 예시는 DescribeTable의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeTable](#)을 참조하세요.

DescribeTimeToLive

다음 코드 예시는 DescribeTimeToLive의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript를 사용하여 기존 DynamoDB 테이블의 TTL 구성을 설명합니다.

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

export const describeTTL = async (tableName, region) => {
  const client = new DynamoDBClient({
    region: region,
```

```

    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  try {
    const ttlDescription = await client.send(new
DescribeTimeToLiveCommand({ TableName: tableName }));

    if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED') {
      console.log("TTL is enabled for table %s.", tableName);
    } else {
      console.log("TTL is not enabled for table %s.", tableName);
    }

    return ttlDescription;
  } catch (e) {
    console.error(`Error describing table: ${e}`);
    throw e;
  }
}

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeTimeToLive](#)를 참조하세요.

ExecuteStatement

다음 코드 예시는 ExecuteStatement의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ExecuteStatement](#)를 참조하세요.

GetItem

다음 코드 예시는 GetItem의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 [GetCommand](#)를 참조하세요.

```
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 참조의 [GetItem](#)을 참조하세요.

ListTables

다음 코드 예시는 ListTables의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListTables](#)를 참조하세요.

PutItem

다음 코드 예시는 PutItem의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 [PutCommand](#)를 참조하세요.

```
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 참조의 [PutItem](#)을 참조하세요.

Query

다음 코드 예시는 Query의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 [QueryCommand](#)를 참조하세요.

```
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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

Scan

다음 코드 예시는 Scan의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 [ScanCommand](#)를 참조하세요.

```
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 [Scan](#)을 참조하세요.

UpdateItem

다음 코드 예시는 UpdateItem의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 [UpdateCommand](#)를 참조하세요.

```
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 참조의 [UpdateItem](#)을 참조하세요.

UpdateTimeToLive

다음 코드 예시는 UpdateTimeToLive의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

기존 DynamoDB 테이블에서 TTL을 활성화합니다.

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

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') => {

  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const params = {
    TableName: tableName,
    TimeToLiveSpecification: {
      Enabled: true,
      AttributeName: ttlAttribute
    }
  };

  try {
    const response = await client.send(new UpdateTimeToLiveCommand(params));
    if (response.$metadata.httpStatusCode === 200) {
      console.log(`TTL enabled successfully for table ${tableName}, using
attribute name ${ttlAttribute}.`);
    } else {
      console.log(`Failed to enable TTL for table ${tableName}, response
object: ${response}`);
    }
    return response;
  } catch (e) {
    console.error(`Error enabling TTL: ${e}`);
    throw e;
  }
};

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

기존 DynamoDB 테이블에서 TTL을 비활성화합니다.

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

```
export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1') => {

  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const params = {
    TableName: tableName,
    TimeToLiveSpecification: {
      Enabled: false,
      AttributeName: ttlAttribute
    }
  };

  try {
    const response = await client.send(new UpdateTimeToLiveCommand(params));
    if (response.$metadata.httpStatusCode === 200) {
      console.log(`TTL disabled successfully for table ${tableName}, using
attribute name ${ttlAttribute}.`);
    } else {
      console.log(`Failed to disable TTL for table ${tableName}, response
object: ${response}`);
    }
    return response;
  } catch (e) {
    console.error(`Error disabling TTL: ${e}`);
    throw e;
  }
};

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateTimeToLive](#)를 참조하세요.

시나리오

DynamoDB 테이블에 데이터를 제출하기 위한 앱 구축

다음 코드 예제에서는 Amazon DynamoDB 테이블에 데이터를 제출하고 사용자가 테이블을 업데이트 하면 알려주는 애플리케이션을 빌드하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SNS

단일 속성과 여러 값 비교

다음 코드 예제에서는 DynamoDB에서 여러 값을 단일 속성과 비교하는 방법을 보여줍니다.

- IN 연산자를 사용하여 단일 속성과 여러 값을 비교합니다.
- IN 연산자를 여러 OR 조건과 비교합니다.
- IN을 사용할 때 성능 및 표현식 복잡성 측면의 이점을 이해합니다.

SDK for JavaScript(v3)

를 사용하여 여러 값을 단일 속성과 비교합니다 AWS SDK for JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  ScanCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");
```

```

/**
 * Query or scan a DynamoDB table to find items where an attribute matches any value
 * from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 * attribute
 * against multiple possible values, which is more efficient than using multiple OR
 * conditions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against the
 * values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key attribute
 * for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 * query
 * @returns {Promise<Object>} - The response from DynamoDB containing the matching
 * items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the filter expression using the IN operator
  const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
    `:val${index}`}).join(', ')}))`;

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation

```

```
if (partitionKeyName && partitionKeyValue) {
  const keyCondition = `${partitionKeyName} = :partitionKey`;
  expressionAttributeValues[':partitionKey'] = partitionKeyValue;

  // Initialize array to collect all items
  let allItems = [];
  let lastEvaluatedKey;

  // Use pagination to get all results
  do {
    const params = {
      TableName: tableName,
      KeyConditionExpression: keyCondition,
      FilterExpression: filterExpression,
      ExpressionAttributeValues: expressionAttributeValues
    };

    // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous query
    if (lastEvaluatedKey) {
      params.ExclusiveStartKey = lastEvaluatedKey;
    }

    const response = await docClient.send(new QueryCommand(params));

    // Add the items from this page to our collection
    if (response.Items && response.Items.length > 0) {
      allItems = [...allItems, ...response.Items];
    }

    // Get the key for the next page of results
    lastEvaluatedKey = response.LastEvaluatedKey;
  } while (lastEvaluatedKey);

  // Return the complete result
  return {
    Items: allItems,
    Count: allItems.length
  };
} else {
  // Otherwise, perform a scan operation
  // Initialize array to collect all items
  let allItems = [];
  let lastEvaluatedKey;
```

```

// Use pagination to get all results
do {
  const params = {
    TableName: tableName,
    FilterExpression: filterExpression,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
  if (lastEvaluatedKey) {
    params.ExclusiveStartKey = lastEvaluatedKey;
  }

  const response = await docClient.send(new ScanCommand(params));

  // Add the items from this page to our collection
  if (response.Items && response.Items.length > 0) {
    allItems = [...allItems, ...response.Items];
  }

  // Get the key for the next page of results
  lastEvaluatedKey = response.LastEvaluatedKey;
} while (lastEvaluatedKey);

// Return the complete result
return {
  Items: allItems,
  Count: allItems.length
};
}
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 * operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 * preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against the
 * values list

```



```

* @param {Array} valuesList - List of values to compare the attribute against
* @param {string} [partitionKeyName] - Optional name of the partition key attribute
for query operations
* @param {string} [partitionKeyValue] - Optional value of the partition key to
query
* @returns {Promise<Object>} - The response from DynamoDB containing the matching
items
*/
async function compareWithOrConditions(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }

  // Create the filter expression using multiple OR conditions
  const filterConditions = valuesList.map((_, index) => `${attributeName} = :val${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;
  }
}

```

```
// Initialize array to collect all items
let allItems = [];
let lastEvaluatedKey;

// Use pagination to get all results
do {
  const params = {
    TableName: tableName,
    KeyConditionExpression: keyCondition,
    FilterExpression: filterExpression,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous query
  if (lastEvaluatedKey) {
    params.ExclusiveStartKey = lastEvaluatedKey;
  }

  const response = await docClient.send(new QueryCommand(params));

  // Add the items from this page to our collection
  if (response.Items && response.Items.length > 0) {
    allItems = [...allItems, ...response.Items];
  }

  // Get the key for the next page of results
  lastEvaluatedKey = response.LastEvaluatedKey;
} while (lastEvaluatedKey);

// Return the complete result
return {
  Items: allItems,
  Count: allItems.length
};
} else {
  // Otherwise, perform a scan operation
  // Initialize array to collect all items
  let allItems = [];
  let lastEvaluatedKey;

  // Use pagination to get all results
  do {
    const params = {
      TableName: tableName,
```

```

        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
    };

    // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
    if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
    }

    const response = await docClient.send(new ScanCommand(params));

    // Add the items from this page to our collection
    if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
    }

    // Get the key for the next page of results
    lastEvaluatedKey = response.LastEvaluatedKey;
} while (lastEvaluatedKey);

// Return the complete result
return {
    Items: allItems,
    Count: allItems.length
};
}
}

```

여러 값을와 비교하는 예제 사용 AWS SDK for JavaScript.

```

/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
    // Example parameters
    const config = { region: "us-west-2" };
    const tableName = "Products";
    const attributeName = "Category";
    const valuesList = ["Electronics", "Computers", "Accessories"];

    console.log(`Searching for products in any of these categories:
    ${valuesList.join(', ')}`);
}

```

```
try {
  // Using the IN operator (recommended approach)
  console.log("\nApproach 1: Using the IN operator");
  const response = await compareMultipleValues(
    config,
    tableName,
    attributeName,
    valuesList
  );

  console.log(`Found ${response.Count} products in the specified categories`);

  // Using multiple OR conditions (alternative approach)
  console.log("\nApproach 2: Using multiple OR conditions");
  const response2 = await compareWithOrConditions(
    config,
    tableName,
    attributeName,
    valuesList
  );

  console.log(`Found ${response2.Count} products in the specified categories`);

  // Example with a query operation
  console.log("\nQuerying a specific manufacturer's products in multiple categories");
  const partitionKeyName = "Manufacturer";
  const partitionKeyValue = "Acme";

  const response3 = await compareMultipleValues(
    config,
    tableName,
    attributeName,
    valuesList,
    partitionKeyName,
    partitionKeyValue
  );

  console.log(`Found ${response3.Count} Acme products in the specified categories`);

  // Explain the benefits of using the IN operator
  console.log("\nBenefits of using the IN operator:");
```

```

    console.log("1. More concise expression compared to multiple OR conditions");
    console.log("2. Better readability and maintainability");
    console.log("3. Potentially better performance with large value lists");
    console.log("4. Simpler code that's less prone to errors");
    console.log("5. Easier to modify when adding or removing values");

  } catch (error) {
    console.error("Error:", error);
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [Query](#)
 - [Scan](#)

항목의 TTL을 조건부로 업데이트

다음 코드 예제에서는 항목의 TTL을 조건부로 업데이트하는 방법을 보여줍니다.

SDK for JavaScript(v3)

조건을 사용하여 테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```

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

export const updateItemConditional = async (tableName, partitionKey, sortKey, region = 'us-east-1', newAttribute = 'default-value') => {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);

  const params = {
    TableName: tableName,
    Key: marshall({
      artist: partitionKey,
      album: sortKey
    }),

```

```

    UpdateExpression: "SET newAttribute = :newAttribute",
    ConditionExpression: "expireAt > :expiration",
    ExpressionAttributeValues: marshall({
      ':newAttribute': newAttribute,
      ':expiration': currentTime
    }),
    ReturnValues: "ALL_NEW"
  });

  try {
    const response = await client.send(new UpdateItemCommand(params));
    const responseData = unmarshall(response.Attributes);
    console.log("Item updated successfully: ", responseData);
    return responseData;
  } catch (error) {
    if (error.name === "ConditionalCheckFailedException") {
      console.log("Condition check failed: Item's 'expireAt' is expired.");
    } else {
      console.error("Error updating item: ", error);
    }
    throw error;
  }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-sort-key-value');

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateItem](#)을 참조하세요.

표현식 연산자 수 계산

다음 코드 예제에서는 DynamoDB에서 표현식 연산자 수를 계산하는 방법을 보여줍니다.

- DynamoDB의 연산자 제한(300개)을 이해합니다.
- 복잡한 표현식에서 연산자 수를 계산합니다.
- 제한 범위를 벗어나지 않도록 표현식을 최적화합니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript를 사용한 표현식 연산자 수 계산을 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 * conditions
 * @returns {Object} - Object containing the filter expression and attribute values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};

  // Generate the specified number of conditions
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        condition = `attribute${i} < ${valueKey}`;
```

```

        expressionAttributeValues[valueKey] = i * 10;
        break;
    case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
    case 4:
        condition = `attribute_exists(attribute${i})`;
        break;
    }

    conditions.push(condition);
}

// Join the conditions with AND or OR
const operator = useAnd ? " AND " : " OR ";
const filterExpression = conditions.join(operator);

// Calculate the operator count
// Each condition has 1 operator (=, >, <, contains, attribute_exists)
// Each AND or OR between conditions is 1 operator
const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount - 1 : 0);

return {
    filterExpression,
    expressionAttributeValues,
    operatorCount
};
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute values
 */
function createComplexUpdateExpression(operationsCount) {
    // Initialize the expression parts and attribute values
    const setOperations = [];
    const expressionAttributeValues = {};

```



```

// Generate the specified number of SET operations
for (let i = 0; i < operationsCount; i++) {
  // Alternate between different types of SET operations
  let operation;
  const valueKey = `:val${i}`;

  switch (i % 3) {
    case 0:
      // Simple assignment (1 operator: =)
      operation = `attribute${i} = ${valueKey}`;
      expressionAttributeValues[valueKey] = `value${i}`;
      break;
    case 1:
      // Addition (2 operators: = and +)
      operation = `attribute${i} = attribute${i} + ${valueKey}`;
      expressionAttributeValues[valueKey] = i;
      break;
    case 2:
      // Conditional assignment with if_not_exists (2 operators: = and
if_not_exists)
      operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
      expressionAttributeValues[valueKey] = i * 10;
      break;
  }

  setOperations.push(operation);
}

// Create the update expression
const updateExpression = `SET ${setOperations.join(", ")}`;

// Calculate the operator count
// Each operation has 1-2 operators as noted above
let operatorCount = 0;
for (let i = 0; i < operationsCount; i++) {
  operatorCount += (i % 3 === 0) ? 1 : 2;
}

return {
  updateExpression,
  expressionAttributeValues,
  operatorCount
};

```

```

}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
async function testOperatorLimit(
  config,
  tableName,
  key,
  operatorCount
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create a complex update expression with the specified operator count
  const { updateExpression, expressionAttributeValues, operatorCount: actualCount } =
    createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to get
    close to target count

  console.log(`Generated update expression with approximately ${actualCount}
    operators`);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Attempt the update operation

```

```

    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      message: `Operation succeeded with ${actualCount} operators`,
      data: response
    };
  } catch (error) {
    // Check if the error is due to exceeding the operator limit
    if (error.name === "ValidationException" &&
        error.message.includes("too many operators")) {
      return {
        success: false,
        message: `Operation failed: ${error.message}`,
        operatorCount: actualCount
      };
    }

    // Return other errors
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      error
    };
  }
}

/**
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
  config,
  tableName,
  key,
  totalOperations
) {

```

```
// Initialize the DynamoDB client
const client = new DynamoDBClient(config);
const docClient = DynamoDBDocumentClient.from(client);

// Calculate how many operations we can safely include in each batch
// Using 150 as a conservative limit (well below 300)
const operationsPerBatch = 100;
const batchCount = Math.ceil(totalOperations / operationsPerBatch);

console.log(`Breaking down ${totalOperations} operations into ${batchCount}
batches`);

const results = [];

// Process each batch
for (let batch = 0; batch < batchCount; batch++) {
  // Calculate the operations for this batch
  const batchStart = batch * operationsPerBatch;
  const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
  const batchSize = batchEnd - batchStart;

  console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
operations`);

  // Create an update expression for this batch
  const { updateExpression, expressionAttributeValues, operatorCount } =
    createComplexUpdateExpression(batchSize);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation for this batch
    const response = await docClient.send(new UpdateCommand(params));

    results.push({
      batch: batch + 1,
      success: true,
    });
  }
}
```



```

const comparisonMatches = expression.match(comparisonRegex) || [];
counts.comparisonOperators = comparisonMatches.length;

// Count logical operators (AND, OR, NOT)
const andMatches = expression.match(/\bAND\b/g) || [];
const orMatches = expression.match(/\bOR\b/g) || [];
const notMatches = expression.match(/\bNOT\b/g) || [];
counts.logicalOperators = andMatches.length + orMatches.length +
notMatches.length;

// Count functions (attribute_exists, attribute_not_exists, attribute_type,
begins_with, contains, size)
const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
const functionMatches = expression.match(functionRegex) || [];
counts.functions = functionMatches.length;

// Count arithmetic operators (+ and -)
const arithmeticMatches = expression.match(/[a-zA-Z0-9_]\s*\[+\-]\s*[a-zA-
Z0-9_(:]/g) || [];
counts.arithmeticOperators = arithmeticMatches.length;

// Count special operators (BETWEEN, IN)
const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
const inMatches = expression.match(/\bIN\b/g) || [];
counts.specialOperators = betweenMatches.length + inMatches.length;

// Add extra operators for BETWEEN (each BETWEEN includes an AND)
counts.logicalOperators += betweenMatches.length;

// Calculate total
counts.total = counts.comparisonOperators +
               counts.logicalOperators +
               counts.functions +
               counts.arithmeticOperators +
               counts.specialOperators;

return counts;
}

```

에서 계산하는 표현식 연산자의 사용 예입니다 AWS SDK for JavaScript.

```

/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating DynamoDB expression operator counting and the 300
operator limit");

  try {
    // Example 1: Analyze a simple expression
    console.log("\nExample 1: Analyzing a simple expression");
    const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
(:cat1, :cat2, :cat3)";
    const simpleCount = countOperatorsInExpression(simpleExpression);

    console.log(`Expression: ${simpleExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
    console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
    console.log(`- Functions: ${simpleCount.functions}`);
    console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
    console.log(`- Special operators: ${simpleCount.specialOperators}`);
    console.log(`- Total operators: ${simpleCount.total}`);

    // Example 2: Analyze a complex expression
    console.log("\nExample 2: Analyzing a complex expression");
    const complexExpression =
      "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
      "(Price > :price AND contains(Description, :keyword) AND " +
      "(Rating >= :minRating OR Reviews > :minReviews))";
    const complexCount = countOperatorsInExpression(complexExpression);

    console.log(`Expression: ${complexExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
    console.log(`- Logical operators: ${complexCount.logicalOperators}`);
    console.log(`- Functions: ${complexCount.functions}`);
    console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
    console.log(`- Special operators: ${complexCount.specialOperators}`);
  }
}

```

```
console.log(`- Total operators: ${complexCount.total}`);

// Example 3: Test approaching the operator limit
console.log("\nExample 3: Testing an expression approaching the operator
limit");
const approachingLimit = await testOperatorLimit(config, tableName, key, 290);
console.log(approachingLimit.message);

// Example 4: Test exceeding the operator limit
console.log("\nExample 4: Testing an expression exceeding the operator limit");
const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
console.log(exceedingLimit.message);

// Example 5: Breaking down a complex expression
console.log("\nExample 5: Breaking down a complex expression into multiple
operations");
const breakdownResult = await breakDownComplexExpression(config, tableName, key,
500);
console.log(`Processed ${breakdownResult.results.length} of
${breakdownResult.totalBatches} batches`);

// Explain the operator counting rules
console.log("\nKey points about DynamoDB expression operator counting:");
console.log("1. The maximum number of operators in any expression is 300");
console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
operator");
console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
console.log("4. Each function call (attribute_exists, contains, etc.) counts as
1 operator");
console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND within
it)");
console.log("7. IN counts as 1 operator regardless of the number of values");
console.log("8. Parentheses for grouping and attribute paths don't count as
operators");
console.log("9. When you exceed the limit, the error always reports '301
operators'");
console.log("10. For complex operations, break them into multiple smaller
operations");

} catch (error) {
  console.error("Error:", error);
}
}
```


- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateItem](#)을 참조하세요.

사진을 관리하기 위한 서버리스 애플리케이션 만들기

다음 코드 예시에서는 사용자가 레이블을 사용하여 사진을 관리할 수 있는 서버리스 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 [AWS 커뮤니티](#)의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

웜 처리량이 활성화된 테이블 만들기

다음 코드 예제에서는 웜 처리량이 활성화된 테이블을 만드는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript를 사용하여 웜 처리량 설정이 있는 DynamoDB 테이블을 만듭니다.

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

export async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
```

```
miscKeyAttr,
nonKeyAttr,
tableProvisionedReadUnits,
tableProvisionedWriteUnits,
tableWarmReads,
tableWarmWrites,
indexName,
indexProvisionedReadUnits,
indexProvisionedWriteUnits,
indexWarmReads,
indexWarmWrites,
region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
        { AttributeName: partitionKey, AttributeType: "S" },
        { AttributeName: sortKey, AttributeType: "S" },
        { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
        { AttributeName: partitionKey, KeyType: "HASH" },
        { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
        ReadCapacityUnits: tableProvisionedReadUnits,
        WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
        ReadUnitsPerSecond: tableWarmReads,
        WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
        {
          IndexName: indexName,
          KeySchema: [
            { AttributeName: sortKey, KeyType: "HASH" },
            { AttributeName: miscKeyAttr, KeyType: "RANGE" },
          ],
          Projection: {
            ProjectionType: "INCLUDE",
            NonKeyAttributes: [nonKeyAttr],
          },
        },
      ],
    });
    await ddbClient.send(command);
  } catch (err) {
    console.error(err);
  }
}
```

```

        },
        ProvisionedThroughput: {
            ReadCapacityUnits: indexProvisionedReadUnits,
            WriteCapacityUnits: indexProvisionedWriteUnits,
        },
        WarmThroughput: {
            ReadUnitsPerSecond: indexWarmReads,
            WriteUnitsPerSecond: indexWarmWrites,
        },
    },
    ],
});
const response = await ddbClient.send(command);
console.log(response);
return response;
} catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
}
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
    'example-table',
    'pk',
    'sk',
    'gsiKey',
    'data',
    10, 10, 5, 5,
    'example-index',
    5, 5, 2, 2
);
*/

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateTable](#)을 참조하세요.

TTL을 사용하여 항목 생성

다음 코드 예제에서는 TTL을 사용해 항목을 만드는 방법을 보여줍니다.

SDK for JavaScript(v3)

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

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  // Get the current time in epoch second format
  const current_time = Math.floor(new Date().getTime() / 1000);

  // Calculate the expireAt time (90 days from now) in epoch second format
  const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 * 1000) /
1000);

  // Create DynamoDB item
  const item = {
    'partitionKey': {'S': partition_key},
    'sortKey': {'S': sort_key},
    'createdAt': {'N': current_time.toString()},
    'expireAt': {'N': expire_at.toString()}
  };

  const putItemCommand = new PutItemCommand({
    TableName: table_name,
    Item: item,
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  client.send(putItemCommand, function(err, data) {
    if (err) {
      console.log("Exception encountered when creating item %s, here's what
happened: ", data, err);
      throw err;
    } else {
      console.log("Item created successfully: %s.", data);
      return data;
    }
  });
}
```

```

}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
  'your-sort-key-value');

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutItem](#)을 참조하세요.

PartiQL DELETE를 사용하여 데이터 삭제

다음 코드 예제에서는 PartiQL DELETE 문을 사용하여 데이터를 삭제하는 방법을 보여줍니다.

SDK for JavaScript(v3)

PartiQL DELETE 문들과 함께 사용하여 DynamoDB 테이블에서 항목을 삭제합니다 AWS SDK for JavaScript.

```

/**
 * This example demonstrates how to delete items from a DynamoDB table using
 * PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {

```

```

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

const params = {
  Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
  Parameters: [partitionKeyValue],
};

try {
  const data = await docClient.send(new ExecuteStatementCommand(params));
  console.log("Item deleted successfully");
  return data;
} catch (err) {
  console.error("Error deleting item:", err);
  throw err;
}
};

/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

```

```

    try {
      const data = await docClient.send(new ExecuteStatementCommand(params));
      console.log("Item deleted successfully");
      return data;
    } catch (err) {
      console.error("Error deleting item:", err);
      throw err;
    }
  };

  /**
   * Delete an item with a condition to ensure the delete only happens if a condition
   * is met.
   *
   * @param tableName - The name of the DynamoDB table
   * @param partitionKeyName - The name of the partition key attribute
   * @param partitionKeyValue - The value of the partition key
   * @param conditionAttribute - The attribute to check in the condition
   * @param conditionValue - The value to compare against in the condition
   * @returns The response from the ExecuteStatementCommand
   */
  export const deleteItemWithCondition = async (
    tableName: string,
    partitionKeyName: string,
    partitionKeyValue: string | number,
    conditionAttribute: string,
    conditionValue: any
  ) => {
    const client = new DynamoDBClient({});
    const docClient = DynamoDBDocumentClient.from(client);

    const params = {
      Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
${conditionAttribute} = ?`,
      Parameters: [partitionKeyValue, conditionValue],
    };

    try {
      const data = await docClient.send(new ExecuteStatementCommand(params));
      console.log("Item deleted with condition successfully");
      return data;
    } catch (err) {
      console.error("Error deleting item with condition:", err);
      throw err;
    }
  };

```

```

    }
  };

  /**
   * Batch delete multiple items using PartiQL.
   *
   * @param tableName - The name of the DynamoDB table
   * @param keys - Array of objects containing key information
   * @returns The response from the BatchExecuteStatementCommand
   */
  export const batchDeleteItems = async (
    tableName: string,
    keys: Array<{
      partitionKeyName: string;
      partitionKeyValue: string | number;
      sortKeyName?: string;
      sortKeyValue?: string | number;
    }>
  ) => {
    const client = new DynamoDBClient({});
    const docClient = DynamoDBDocumentClient.from(client);

    // Create statements for each delete
    const statements = keys.map((key) => {
      if (key.sortKeyName && key.sortKeyValue !== undefined) {
        return {
          Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ? AND
${key.sortKeyName} = ?`,
          Parameters: [key.partitionKeyValue, key.sortKeyValue],
        };
      } else {
        return {
          Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?`,
          Parameters: [key.partitionKeyValue],
        };
      }
    });

    const params = {
      Statements: statements,
    };

    try {
      const data = await docClient.send(new BatchExecuteStatementCommand(params));
    }
  };

```



```

        console.log("Items batch deleted successfully");
        return data;
    } catch (err) {
        console.error("Error batch deleting items:", err);
        throw err;
    }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemsByFilter = async (
    tableName: string,
    filterAttribute: string,
    filterValue: any
) => {
    const client = new DynamoDBClient({});
    const docClient = DynamoDBDocumentClient.from(client);

    const params = {
        Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
        Parameters: [filterValue],
    };

    try {
        const data = await docClient.send(new ExecuteStatementCommand(params));
        console.log("Items deleted by filter successfully");
        return data;
    } catch (err) {
        console.error("Error deleting items by filter:", err);
        throw err;
    }
};

/**
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {

```

```
// Delete an item by partition key (simple primary key)
await deleteItemByPartitionKey("UsersTable", "userId", "user123");

// Delete an item by composite key (partition key + sort key)
await deleteItemByCompositeKey(
  "OrdersTable",
  "orderId",
  "order456",
  "productId",
  "prod789"
);

// Delete with a condition
await deleteItemWithCondition(
  "UsersTable",
  "userId",
  "user789",
  "userStatus",
  "inactive"
);

// Batch delete multiple items
await batchDeleteItems("UsersTable", [
  { partitionKeyName: "userId", partitionKeyValue: "user234" },
  { partitionKeyName: "userId", partitionKeyValue: "user345" },
]);

// Batch delete items with composite keys
await batchDeleteItems("OrdersTable", [
  {
    partitionKeyName: "orderId",
    partitionKeyValue: "order567",
    sortKeyName: "productId",
    sortKeyValue: "prod123",
  },
  {
    partitionKeyName: "orderId",
    partitionKeyValue: "order678",
    sortKeyName: "productId",
    sortKeyValue: "prod456",
  },
]);

// Delete items by filter (use with caution)
```

```
await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [BatchExecuteStatement](#)
 - [ExecuteStatement](#)

PartiQL INSERT를 사용하여 데이터 삽입

다음 코드 예제에서는 PartiQL INSERT 문을 사용하여 데이터를 삽입하는 방법을 보여줍니다.

SDK for JavaScript(v3)

와 함께 PartiQL INSERT 문을 사용하여 DynamoDB 테이블에 항목을 삽입합니다 AWS SDK for JavaScript.

```
/**
 * This example demonstrates how to insert items into a DynamoDB table using
 * PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItem = async (tableName: string, item: Record<string, any>) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Convert the item to a string representation for PartiQL
```

```

const itemString = JSON.stringify(item).replace(/"([^\"]+)"/g, '$1:');

const params = {
  Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
};

try {
  const data = await docClient.send(new ExecuteStatementCommand(params));
  console.log("Item inserted successfully");
  return data;
} catch (err) {
  console.error("Error inserting item:", err);
  throw err;
}
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string,
any>[]) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([^\"]+)"/g, '$1:');
    return {
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
  }
};

```

```

        return data;
    } catch (err) {
        console.error("Error batch inserting items:", err);
        throw err;
    }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
    tableName: string,
    item: Record<string, any>,
    partitionKeyName: string
) => {
    const client = new DynamoDBClient({});
    const docClient = DynamoDBDocumentClient.from(client);

    const itemString = JSON.stringify(item).replace(/"([^\"]+)":/g, '$1:');
    const partitionKeyValue = JSON.stringify(item[partitionKeyName]);

    const params = {
        Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
attribute_not_exists(${partitionKeyName})`,
        Parameters: [{ S: partitionKeyValue }],
    };

    try {
        const data = await docClient.send(new ExecuteStatementCommand(params));
        console.log("Item inserted with condition successfully");
        return data;
    } catch (err) {
        console.error("Error inserting item with condition:", err);
        throw err;
    }
};

/**
```

```
* Example usage showing how to insert items with different index types
*/
export const insertExamples = async () => {
  // Example table with a simple primary key (just partition key)
  const simpleKeyItem = {
    userId: "user123",
    name: "John Doe",
    email: "john@example.com",
  };
  await insertItem("UsersTable", simpleKeyItem);

  // Example table with composite key (partition key + sort key)
  const compositeKeyItem = {
    orderId: "order456",
    productId: "prod789",
    quantity: 2,
    price: 29.99,
  };
  await insertItem("OrdersTable", compositeKeyItem);

  // Example with Global Secondary Index (GSI)
  // The GSI might be on the email attribute
  const gsiItem = {
    userId: "user789",
    email: "jane@example.com",
    name: "Jane Smith",
    userType: "premium", // This could be part of a GSI
  };
  await insertItem("UsersTable", gsiItem);

  // Example with Local Secondary Index (LSI)
  // LSI uses the same partition key but different sort key
  const lsiItem = {
    orderId: "order567", // Partition key
    productId: "prod123", // Sort key for the table
    orderDate: "2023-11-15", // Potential sort key for an LSI
    quantity: 1,
    price: 19.99,
  };
  await insertItem("OrdersTable", lsiItem);

  // Batch insert example with multiple items
  const batchItems = [
    {
```

```
    userId: "user234",
    name: "Alice Johnson",
    email: "alice@example.com",
  },
  {
    userId: "user345",
    name: "Bob Williams",
    email: "bob@example.com",
  },
];
await batchInsertItems("UsersTable", batchItems);
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [BatchExecuteStatement](#)
 - [ExecuteStatement](#)

브라우저에서 Lambda 함수 간접 호출

다음 코드 예제에서는 브라우저에서 AWS Lambda 함수를 호출하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS Lambda 함수를 사용하여 사용자 선택 항목으로 Amazon DynamoDB 테이블을 업데이트하는 브라우저 기반 애플리케이션을 생성할 수 있습니다. 이 앱은 AWS SDK for JavaScript v3를 사용합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Lambda

고급 쿼리 작업 수행

다음 코드 예제에서는 DynamoDB에서 고급 쿼리 작업을 수행하는 방법을 보여줍니다.

- 다양한 필터링 및 조건 기법을 사용하여 테이블을 쿼리합니다.

- 대규모 결과 세트에 페이지 매김을 구현합니다.
- 대체 액세스 패턴에 글로벌 보조 인덱스를 사용합니다.
- 애플리케이션 요구 사항에 따라 일관성 제어를 적용합니다.

SDK for JavaScript(v3)

를 사용하여 강력히 일관된 읽기로 쿼리합니다 AWS SDK for JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };
  }
}
```



```

    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}

```

와 함께 글로벌 보조 인덱스를 사용하여 쿼리합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query

```

```

        const command = new QueryCommand(input);
        return await client.send(command);
    } catch (error) {
        console.error(`Error querying table: ${error}`);
        throw error;
    }
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
    config,
    tableName,
    indexName,
    gameId
) {
    try {
        // Create DynamoDB client
        const client = new DynamoDBClient(config);

        // Construct the query input for the GSI
        const input = {
            TableName: tableName,
            IndexName: indexName,
            KeyConditionExpression: "game_id = :gameId",
            ExpressionAttributeValues: {
                ":gameId": { S: gameId }
            }
        };

        // Execute the query
        const command = new QueryCommand(input);
        return await client.send(command);
    } catch (error) {
        console.error(`Error querying GSI: ${error}`);
        throw error;
    }
}

```

```
}

```

를 사용하여 페이지 매김으로 쿼리합니다 AWS SDK for JavaScript.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 * pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;
  }
}
```

```
// Loop until all pages are retrieved
do {
  // Construct the query input
  const input = {
    TableName: tableName,
    KeyConditionExpression: "#pk = :pkValue",
    Limit: pageSize,
    ExpressionAttributeNames: {
      "#pk": partitionKeyName
    },
    ExpressionAttributeValues: {
      ":pkValue": { S: partitionKeyValue }
    }
  };

  // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous query
  if (lastEvaluatedKey) {
    input.ExclusiveStartKey = lastEvaluatedKey;
  }

  // Execute the query
  const command = new QueryCommand(input);
  const response = await client.send(command);

  // Process the current page of results
  pageCount++;
  console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

  // Add the items from this page to our collection
  if (response.Items && response.Items.length > 0) {
    allItems.push(...response.Items);
  }

  // Get the LastEvaluatedKey for the next page
  lastEvaluatedKey = response.LastEvaluatedKey;

} while (lastEvaluatedKey); // Continue until there are no more pages

console.log(`Query complete. Retrieved ${allItems.length} items in ${pageCount}
pages.`);
return allItems;
} catch (error) {
  console.error(`Error querying with pagination: ${error}`);
}
```

```

        throw error;
    }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };

```

를 사용하여 복잡한 필터로 쿼리합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set

```

```

* @returns {Promise<Object>} - The query response
*/
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}

```

를 사용하여 동적으로 구성된 필터 표현식을 사용하여 쿼리합니다 AWS SDK for JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }
  }
}
```

```

    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

목록 작업 수행

다음 코드 예제에서는 DynamoDB에서 목록 작업을 수행하는 방법을 보여줍니다.

- 목록 속성에 요소를 추가합니다.
- 목록 속성에서 요소를 제거합니다.
- 인덱스별로 목록의 특정 요소를 업데이트합니다.
- 목록 추가 및 목록 인덱스 함수를 사용합니다.

SDK for JavaScript(v3)

를 사용하여 목록 작업을 시연합니다 AWS SDK for JavaScript.


```

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} =
list_append(if_not_exists(${listName}, :empty_list), :values)`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

```

```

};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = list_append(:values,
if_not_exists(${listName}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

```

```

};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${index}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };
};

```

```

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
    config,
    tableName,
    key,
    listName,
    index
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using REMOVE with index notation
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `REMOVE ${listName}[${index}]`,
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

```

```
/**
 * Concatenate two lists.
 *
 * This function demonstrates how to concatenate two lists using the list_append
 function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${resultListName} =
list_append(if_not_exists(${listName1}, :empty_list),
if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));
}
```

```

    return response;
  }

  /**
   * Create a nested list structure.
   *
   * This function demonstrates how to create and work with nested lists.
   *
   * @param {Object} config - AWS configuration object
   * @param {string} tableName - The name of the DynamoDB table
   * @param {Object} key - The key of the item to update
   * @param {string} listName - The name of the list attribute
   * @param {Array} nestedLists - An array of arrays to create a nested list structure
   * @returns {Promise<Object>} - The response from DynamoDB
   */
  async function createNestedList(
    config,
    tableName,
    key,
    listName,
    nestedLists
  ) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters to create a nested list
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `SET ${listName} = :nested_lists`,
      ExpressionAttributeValues: {
        ":nested_lists": nestedLists
      },
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
  }

  /**

```

```
* Update an element in a nested list.
*
* This function demonstrates how to update an element in a nested list
* using multiple index notations.
*
* @param {Object} config - AWS configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The key of the item to update
* @param {string} listName - The name of the list attribute
* @param {number} outerIndex - The index in the outer list
* @param {number} innerIndex - The index in the inner list
* @param {any} value - The new value for the element
* @returns {Promise<Object>} - The response from DynamoDB
*/
async function updateNestedListElement(
  config,
  tableName,
  key,
  listName,
  outerIndex,
  innerIndex,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using multiple index notations
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}
```

```

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}

```

를 사용한 목록 작업 사용의 예입니다 AWS SDK for JavaScript.

```

/**
 * Example of how to work with lists in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

```



```
console.log("Demonstrating list operations in DynamoDB");

try {
  // Example 1: Append elements to a list
  console.log("\nExample 1: Appending elements to a list");
  const response1 = await appendToList(
    config,
    tableName,
    key,
    "RecentSearches",
    ["laptop", "headphones", "monitor"]
  );

  console.log("Appended to list:", response1.Attributes);

  // Example 2: Prepend elements to a list
  console.log("\nExample 2: Prepending elements to a list");
  const response2 = await prependToList(
    config,
    tableName,
    key,
    "RecentSearches",
    ["keyboard", "mouse"]
  );

  console.log("Prepended to list:", response2.Attributes);

  // Get the current state of the item
  let currentItem = await getItem(config, tableName, key);
  console.log("\nCurrent state of RecentSearches:", currentItem?.RecentSearches);

  // Example 3: Update a specific element in a list
  console.log("\nExample 3: Updating a specific element in a list");
  const response3 = await updateListElement(
    config,
    tableName,
    key,
    "RecentSearches",
    0, // Update the first element
    "mechanical keyboard" // New value
  );

  console.log("Updated list element:", response3.Attributes);
}
```

```
// Example 4: Remove an element from a list
console.log("\nExample 4: Removing an element from a list");
const response4 = await removeListElement(
    config,
    tableName,
    key,
    "RecentSearches",
    2 // Remove the third element
);

console.log("List after removing element:", response4.Attributes);

// Example 5: Create and concatenate lists
console.log("\nExample 5: Creating and concatenating lists");

// First, create two separate lists
await updateWithMultipleActions(
    config,
    tableName,
    key,
    "SET WishList = :wishlist, SavedItems = :saveditems",
    null,
    {
        ":wishlist": ["gaming laptop", "wireless earbuds"],
        ":saveditems": ["smartphone", "tablet"]
    }
);

// Then, concatenate them
const response5 = await concatenateLists(
    config,
    tableName,
    key,
    "WishList",
    "SavedItems",
    "AllItems"
);

console.log("Concatenated lists:", response5.Attributes);

// Example 6: Create a nested list structure
console.log("\nExample 6: Creating a nested list structure");
const response6 = await createNestedList(
```

```
    config,
    tableName,
    key,
    "Categories",
    [
      ["Electronics", "Computers", "Accessories"],
      ["Books", "Magazines", "E-books"],
      ["Clothing", "Shoes", "Watches"]
    ]
  );

console.log("Created nested list:", response6.Attributes);

// Example 7: Update an element in a nested list
console.log("\nExample 7: Updating an element in a nested list");
const response7 = await updateNestedListElement(
  config,
  tableName,
  key,
  "Categories",
  0, // First inner list
  1, // Second element in that list
  "Laptops" // New value
);

console.log("Updated nested list element:", response7.Attributes);

// Get the final state of the item
currentItem = await getItem(config, tableName, key);
console.log("\nFinal state of the item:", JSON.stringify(currentItem, null, 2));

// Explain list operations
console.log("\nKey points about list operations in DynamoDB:");
console.log("1. Use list_append to add elements to a list");
console.log("2. To append elements, use list_append(existingList, newElements)");
console.log("3. To prepend elements, use list_append(newElements, existingList)");
console.log("4. Use if_not_exists to handle cases where the list might not exist yet");
console.log("5. Use index notation (list[0]) to access or update specific elements");
console.log("6. Use REMOVE with index notation to remove elements from a list");
console.log("7. Lists can contain elements of different types");
```

```

        console.log("8. Lists can be nested (lists of lists)");
        console.log("9. Use multiple index notations (list[0][1]) to access nested list
elements");

    } catch (error) {
        console.error("Error:", error);
    }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
    config,
    tableName,
    key,
    updateExpression,
    expressionAttributeNames,
    expressionAttributeValues
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Prepare the update parameters
    const updateParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: updateExpression,
        ReturnValues: "UPDATED_NEW"
    };

    // Add expression attribute names if provided
    if (expressionAttributeNames) {
        updateParams.ExpressionAttributeNames = expressionAttributeNames;
    }
}

```

```

    }

    // Add expression attribute values if provided
    if (expressionAttributeValues) {
        updateParams.ExpressionAttributeValues = expressionAttributeValues;
    }

    // Execute the update
    const response = await docClient.send(new UpdateCommand(updateParams));

    return response;
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateItem](#)을 참조하세요.

맵 작업 수행

다음 코드 예제에서는 DynamoDB에서 맵 작업을 수행하는 방법을 보여줍니다.

- 맵 구조에서 중첩 속성을 추가하고 업데이트합니다.
- 맵에서 특정 필드를 제거합니다.
- 깊이 중첩된 맵 속성을 사용하여 작업합니다.

SDK for JavaScript(v3)

를 사용하여 맵 작업을 시연합니다 AWS SDK for JavaScript.

```

/**
 * Example of updating map attributes in DynamoDB.
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update scenarios.
 */

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
    DynamoDBDocumentClient,
    UpdateCommand,
    GetCommand
}

```

```

} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeSafe(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {

```

```

    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update expression
is invalid")) {

        // Create the map with the specified key-value pair
        const createParams = {
            TableName: tableName,
            Key: key,
            UpdateExpression: `SET ${mapName} = :map`,
            ExpressionAttributeValues: {
                ":map": { [mapKey]: value }
            },
            ReturnValues: "UPDATED_NEW"
        };

        return await docClient.send(new UpdateCommand(createParams));
    }

    // Re-throw other errors
    throw error;
}
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeWithIfNotExists(
    config,
    tableName,
    key,
    mapName,
    mapKey,

```

```

    value
  ) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using SET with if_not_exists
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
${mapName}.${mapKey} = :value`,
      ExpressionAttributeValues: {
        ":emptyMap": {},
        ":value": value
      },
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
  }

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} path - The path to the nested attribute as an array of keys
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value

```



```

) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
      updateExpression += ` ${parentPath} = if_not_exists(${parentPath}, :emptyMap
${i}),`;
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[`:value`] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 */

```

```

* This function demonstrates how to update multiple fields in a map
* in a single DynamoDB operation.
*
* @param {Object} config - AWS configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The key of the item to update
* @param {string} mapName - The name of the map attribute
* @param {Object} updates - Object containing key-value pairs to update
* @returns {Promise<Object>} - The response from DynamoDB
*/
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
  updates
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
    expressionAttributeValues[` :val${index}`] = value;
  });

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

```

```
    return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
    config,
    tableName,
    key
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the get parameters
    const params = {
        TableName: tableName,
        Key: key
    };

    // Perform the get operation
    const response = await docClient.send(new GetCommand(params));

    // Return the item if it exists, otherwise null
    return response.Item || null;
}

/**
 * Example of how to use the map attribute update functions.
 */
async function exampleUsage() {
    // Example parameters
    const config = { region: "us-west-2" };
    const tableName = "Users";
    const key = { UserId: "U12345" };
```

```
console.log("Demonstrating different approaches to update map attributes in
DynamoDB");

try {
  // Example 1: Update a map attribute that might not exist (two-step approach)
  console.log("\nExample 1: Updating a map attribute that might not exist (two-
step approach)");
  const response1 = await updateMapAttributeSafe(
    config,
    tableName,
    key,
    "Preferences",
    "Theme",
    "Dark"
  );

  console.log("Updated preferences:", response1.Attributes);

  // Example 2: Update a map attribute using if_not_exists (elegant approach)
  console.log("\nExample 2: Updating a map attribute using if_not_exists (elegant
approach)");
  const response2 = await updateMapAttributeWithIfNotExists(
    config,
    tableName,
    key,
    "Settings",
    "NotificationsEnabled",
    true
  );

  console.log("Updated settings:", response2.Attributes);

  // Example 3: Update a deeply nested attribute
  console.log("\nExample 3: Updating a deeply nested attribute");
  const response3 = await addToNestedMap(
    config,
    tableName,
    key,
    ["Profile", "Address", "City"],
    "Seattle"
  );

  console.log("Updated nested attribute:", response3.Attributes);
```

```
// Example 4: Update multiple fields in a map
console.log("\nExample 4: Updating multiple fields in a map");
const response4 = await updateMultipleMapFields(
  config,
  tableName,
  key,
  "ContactInfo",
  {
    Email: "user@example.com",
    Phone: "555-123-4567",
    PreferredContact: "Email"
  }
);

console.log("Updated multiple fields:", response4.Attributes);

// Get the final state of the item
console.log("\nFinal state of the item:");
const item = await getItem(config, tableName, key);
console.log(JSON.stringify(item, null, 2));

// Explain the benefits of different approaches
console.log("\nKey points about updating map attributes:");
console.log("1. Use if_not_exists to handle maps that might not exist");
console.log("2. Multiple updates can be combined in a single operation");
console.log("3. Deeply nested attributes require creating parent maps");
console.log("4. DynamoDB expressions are atomic - the entire update succeeds or fails");
console.log("5. Using a single operation is more efficient than multiple separate updates");

} catch (error) {
  console.error("Error:", error);
}

// Export the functions
module.exports = {
  updateMapAttributeSafe,
  updateMapAttributeWithIfNotExists,
  addToNestedMap,
  updateMultipleMapFields,
  getItem,
```

```

    exampleUsage
  };

  // Run the example if this file is executed directly
  if (require.main === module) {
    exampleUsage();
  }

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateItem](#)을 참조하세요.

집합 작업 수행

다음 코드 예제에서는 DynamoDB에서 집합 작업을 수행하는 방법을 보여줍니다.

- 집합 속성에 요소를 추가합니다.
- 집합 속성에서 요소를 제거합니다.
- 집합에 ADD 및 DELETE 작업을 사용합니다.

SDK for JavaScript(v3)

를 사용하여 세트 작업을 시연합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')

```

```
* @returns {Promise<Object>} - The response from DynamoDB
*/
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
```

```

* Remove elements from a set attribute.
*
* This function demonstrates using the DELETE operation to remove elements from a
set.
*
* @param {Object} config - AWS configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The key of the item to update
* @param {string} setName - The name of the set attribute
* @param {Array} values - The values to remove from the set
* @param {string} setType - The type of set ('string', 'number', or 'binary')
* @returns {Promise<Object>} - The response from DynamoDB
*/
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using DELETE
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `DELETE ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    }
  }

```



```

    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {

```

```

    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using SET
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${setName} = :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The new values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function replaceSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}

```

```
/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 * set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
    return {
      success: false,
      message: "Set doesn't exist or is already empty",
      item: currentItem
    };
  }

  // Get the set values
  const setValues = Array.from(currentItem[setName]);

  // If there's only one element left, remove the attribute entirely
  if (setValues.length === 1) {
    // Define the update parameters to remove the attribute
    const params = {
      TableName: tableName,
      Key: key,
```

```

        UpdateExpression: `REMOVE ${setName}`,
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    await docClient.send(new UpdateCommand(params));

    return {
        success: true,
        message: "Last element removed, attribute has been deleted",
        removedValue: setValues[0]
    };
} else {
    // Otherwise, remove just the last element
    // Create a set with just the last element
    const lastElement = setValues[setValues.length - 1];
    const setType = typeof lastElement === 'number' ? 'number' : 'string';

    // Remove the last element
    const response = await removeFromSet(
        config,
        tableName,
        key,
        setName,
        [lastElement],
        setType
    );

    return {
        success: true,
        message: "Last element removed, set still contains elements",
        removedValue: lastElement,
        remainingSet: response.Attributes[setName]
    };
}
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table

```

```

* @param {Object} key - The key of the item to get
* @returns {Promise<Object|null>} - The item or null if not found
*/
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}

```

를 사용한 설정 작업 사용의 예입니다 AWS SDK for JavaScript.

```

/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating set operations in DynamoDB");

  try {
    // Example 1: Create a string set
    console.log("\nExample 1: Creating a string set");
    const response1 = await createSet(

```

```
    config,
    tableName,
    key,
    "Interests",
    ["Reading", "Hiking", "Cooking"],
    "string"
  );

  console.log("Created set:", response1.Attributes);

  // Example 2: Add elements to a set
  console.log("\nExample 2: Adding elements to a set");
  const response2 = await addToSet(
    config,
    tableName,
    key,
    "Interests",
    ["Photography", "Travel"],
    "string"
  );

  console.log("Updated set after adding elements:", response2.Attributes);

  // Example 3: Remove elements from a set
  console.log("\nExample 3: Removing elements from a set");
  const response3 = await removeFromSet(
    config,
    tableName,
    key,
    "Interests",
    ["Cooking"],
    "string"
  );

  console.log("Updated set after removing elements:", response3.Attributes);

  // Example 4: Create a number set
  console.log("\nExample 4: Creating a number set");
  const response4 = await createSet(
    config,
    tableName,
    key,
    "FavoriteNumbers",
    [7, 42, 99],
```

```
    "number"
  );

  console.log("Created number set:", response4.Attributes);

  // Example 5: Replace an entire set
  console.log("\nExample 5: Replacing an entire set");
  const response5 = await replaceSet(
    config,
    tableName,
    key,
    "Interests",
    ["Gaming", "Movies", "Music"],
    "string"
  );

  console.log("Replaced set:", response5.Attributes);

  // Example 6: Remove the last element from a set
  console.log("\nExample 6: Removing the last element from a set");

  // First, create a set with just one element
  await createSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags",
    ["LastTag"],
    "string"
  );

  // Then, remove the last element
  const response6 = await removeLastElementFromSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags"
  );

  console.log(response6.message);
  console.log("Removed value:", response6.removedValue);

  // Get the final state of the items
  console.log("\nFinal state of the items:");
```

```
const item1 = await getItem(config, tableName, key);
console.log("User U12345:", JSON.stringify(item1, null, 2));

const item2 = await getItem(config, tableName, { UserId: "U67890" });
console.log("User U67890:", JSON.stringify(item2, null, 2));

// Explain set operations
console.log("\nKey points about set operations in DynamoDB:");
console.log("1. Use ADD to add elements to a set (duplicates are automatically removed)");
console.log("2. Use DELETE to remove elements from a set");
console.log("3. Use SET to create a new set or replace an existing one");
console.log("4. DynamoDB supports three types of sets: string sets, number sets, and binary sets");
console.log("5. When you delete the last element from a set, the attribute remains as an empty set");
console.log("6. To remove an empty set, use the REMOVE operation");
console.log("7. Sets automatically maintain unique values (no duplicates)");
console.log("8. You cannot mix data types within a set");

} catch (error) {
  console.error("Error:", error);
}
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateItem](#)을 참조하세요.

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.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

배치 PartiQL 명령문을 실행합니다.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

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

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
```

```

    const deleteTable = await input.handle({}, { confirmAll });
    if (deleteTable) {
        await client.send(new DeleteTableCommand({ tableName }));
    } else {
        console.warn(
            "Scenario could not run. Either delete ${tableName} or provide a unique
table name.",
        );
        return;
    }
} catch (caught) {
    if (
        caught instanceof Error &&
        caught.name === "ResourceNotFoundException"
    ) {
        // Do nothing. This means the table is not there.
    } else {
        throw caught;
    }
}

/**
 * Create a table.
 */

log("Creating a table.");
const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
        {
            AttributeName: "name",
            // 'S' is a data type descriptor that represents a number type.
            // For a list of all data type descriptors, see the following link.
            // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
            AttributeType: "S",
        },
    ],
});
// The KeySchema defines the primary key. The primary key can be

```

```

    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/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 참조의 [BatchExecuteStatement](#)를 참조하세요.

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.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

단일 PartiQL 문을 실행합니다.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

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

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
```

```

    "deleteTable",
    `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
    { type: "confirm", confirmAll },
  );
  const deleteTable = await input.handle({});
  if (deleteTable) {
    await client.send(new DeleteTableCommand({ tableName }));
  } else {
    console.warn(
      "Scenario could not run. Either delete ${tableName} or provide a unique
table name.",
    );
    return;
  }
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "ResourceNotFoundException"
  ) {
    // Do nothing. This means the table is not there.
  } else {
    throw caught;
  }
}

/**
 * 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 참조의 [ExecuteStatement](#)를 참조하세요.

글로벌 보조 인덱스를 사용하여 테이블 쿼리

다음 코드 예제에서는 글로벌 보조 인덱스를 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 프라이머리 키를 사용하여 DynamoDB 테이블을 쿼리합니다.
- 글로벌 보조 인덱스(GSI)에서 대체 액세스 패턴을 쿼리합니다.
- 테이블 쿼리와 GSI 쿼리를 비교합니다.

SDK for JavaScript(v3)

와 함께 기본 키를 사용하여 DynamoDB 테이블을 쿼리합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,

```

```

        KeyConditionExpression: "user_id = :userId",
        ExpressionAttributeValues: {
            ":userId": { S: userId }
        }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
} catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
}
}

```

를 사용하여 DynamoDB 글로벌 보조 인덱스(GSI)를 쿼리합니다 AWS SDK for JavaScript.

```

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
    config,
    tableName,
    indexName,
    gameId
) {
    try {
        // Create DynamoDB client
        const client = new DynamoDBClient(config);

        // Construct the query input for the GSI
        const input = {
            TableName: tableName,
            IndexName: indexName,
            KeyConditionExpression: "game_id = :gameId",
            ExpressionAttributeValues: {

```

```

        ":gameId": { S: gameId }
    }
};

// Execute the query
const command = new QueryCommand(input);
return await client.send(command);
} catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
}
}
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

`begins_with` 조건을 사용하여 테이블 쿼리

다음 코드 예제에서는 `begins_with` 조건을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 키 조건 표현식에서 `begins_with` 함수를 사용합니다.
- 정렬 키의 접두사 패턴을 기준으로 항목을 필터링합니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript로 정렬 키에 `begins_with` 조건을 사용하여 DynamoDB 테이블을 쿼리합니다.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 * prefix
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */

```

```

async function queryWithBeginsWith(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with begins_with: ${error}`);
    throw error;
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

날짜 범위를 사용하여 테이블 쿼리

다음 코드 예제에서는 정렬 키에 날짜 범위를 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 특정 날짜 범위 내의 항목을 쿼리합니다.

- 날짜 형식 정렬 키에 비교 연산자를 사용합니다.

SDK for JavaScript(v3)

DynamoDB 테이블에서 날짜 범위 내의 항목을 쿼리합니다 AWS SDK for JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 * attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
AND :endDate',
```

```

    ExpressionAttributeNames: {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    },
    ExpressionAttributeValues: {
      ":pkValue": { S: partitionKeyValue },
      ":startDate": { S: formattedStartDate },
      ":endDate": { S: formattedEndDate }
    }
  };

  // Execute the query
  const command = new QueryCommand(input);
  return await client.send(command);
} catch (error) {
  console.error(`Error querying by date range on sort key: ${error}`);
  throw error;
}
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

복잡한 필터 표현식을 사용하여 테이블 쿼리

다음 코드 예제에서는 복잡한 필터 표현식을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 쿼리 결과에 복잡한 필터 표현식을 적용합니다.
- 논리적 연산자를 사용하여 여러 조건을 결합합니다.
- 키가 아닌 속성을 기준으로 항목을 필터링합니다.

SDK for JavaScript(v3)

를 사용하여 복잡한 필터 표현식을 사용하여 DynamoDB 테이블을 쿼리합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *

```

```

* @param {Object} config - AWS SDK configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {string} partitionKeyName - The name of the partition key
* @param {string} partitionKeyValue - The value of the partition key
* @param {number|string} minViews - Minimum number of views for filtering
* @param {number|string} minReplies - Minimum number of replies for filtering
* @param {string} requiredTag - Tag that must be present in the item's tags set
* @returns {Promise<Object>} - The query response
*/
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
  }
}

```



```

        throw error;
    }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

동적 필터 표현식을 사용하여 테이블 쿼리

다음 코드 예제에서는 동적 필터 표현식을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 런타임에 동적으로 필터 표현식을 작성합니다.
- 사용자 입력 또는 애플리케이션 상태를 기반으로 필터 조건을 구성합니다.
- 조건부로 필터 기준을 추가하거나 제거합니다.

SDK for JavaScript(v3)

를 사용하여 동적으로 구성된 필터 표현식을 사용하여 DynamoDB 테이블을 쿼리합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
  }
}

```

```
const expressionAttributeNames = {
  "#pk": partitionKeyName,
  "#sk": sortKeyName
};

// Add status filter if provided
if (filterParams.status) {
  filterExpressions.push("status = :status");
  expressionAttributeValues[":status"] = { S: filterParams.status };
}

// Add minimum views filter if provided
if (filterParams.minViews !== undefined) {
  filterExpressions.push("views >= :minViews");
  expressionAttributeValues[":minViews"] = { N:
filterParams.minViews.toString() };
}

// Add author filter if provided
if (filterParams.author) {
  filterExpressions.push("author = :author");
  expressionAttributeValues[":author"] = { S: filterParams.author };
}

// Construct the query input
const input = {
  TableName: tableName,
  KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
};

// Add filter expression if any filters were provided
if (filterExpressions.length > 0) {
  input.FilterExpression = filterExpressions.join(" AND ");
}

// Add expression attribute names and values
input.ExpressionAttributeNames = expressionAttributeNames;
input.ExpressionAttributeValues = expressionAttributeValues;

// Execute the query
const command = new QueryCommand(input);
return await client.send(command);
} catch (error) {
  console.error(`Error querying with dynamic filter: ${error}`);
}
```

```

        throw error;
    }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

중첩 속성을 사용하여 테이블 쿼리

다음 코드 예제에서는 중첩 속성을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- DynamoDB 항목의 중첩 속성을 기준으로 액세스하고 필터링합니다.
- 문서 경로 표현식을 사용하여 중첩된 요소를 참조합니다.

SDK for JavaScript(v3)

를 사용하여 중첩 속성으로 DynamoDB 테이블을 쿼리합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
async function queryWithNestedAttribute(
    config,
    tableName,
    productId,
    category
) {
    try {
        // Create DynamoDB client
        const client = new DynamoDBClient(config);

        // Construct the query input
        const input = {
            TableName: tableName,

```

```

    KeyConditionExpression: "product_id = :productId",
    FilterExpression: "details.category = :category",
    ExpressionAttributeValues: {
      ":productId": { S: productId },
      ":category": { S: category }
    }
  };

  // Execute the query
  const command = new QueryCommand(input);
  return await client.send(command);
} catch (error) {
  console.error(`Error querying with nested attribute: ${error}`);
  throw error;
}
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

페이지 매김을 사용하여 테이블 쿼리

다음 코드 예제에서는 페이지 매김을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- DynamoDB 쿼리 결과에 페이지 매김을 구현합니다.
- LastEvaluatedKey를 사용하여 후속 페이지를 검색합니다.
- Limit 파라미터를 사용하여 페이지당 항목 수를 제어합니다.

SDK for JavaScript(v3)

를 사용하여 페이지 매김으로 DynamoDB 테이블을 쿼리합니다 AWS SDK for JavaScript.

```

/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 * pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey

```

```

*/
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };
    };
  }

```

```

    // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous query
    if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
    }

    // Execute the query
    const command = new QueryCommand(input);
    const response = await client.send(command);

    // Process the current page of results
    pageCount++;
    console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

    // Add the items from this page to our collection
    if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
    }

    // Get the LastEvaluatedKey for the next page
    lastEvaluatedKey = response.LastEvaluatedKey;

} while (lastEvaluatedKey); // Continue until there are no more pages

console.log(`Query complete. Retrieved ${allItems.length} items in ${pageCount}
pages.`);
return allItems;
} catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
}
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );

```

```

*
* console.log(`Total items retrieved: ${allItems.length}`);
*
* // Notes on pagination:
* // - LastEvaluatedKey contains the primary key of the last evaluated item
* // - When LastEvaluatedKey is undefined/null, there are no more items to retrieve
* // - ExclusiveStartKey tells DynamoDB where to start the next page
* // - Pagination helps manage memory usage for large result sets
* // - Each page requires a separate network request to DynamoDB
*/

module.exports = { queryWithPagination };

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

강력하게 일관된 읽기를 사용하여 테이블 쿼리

다음 코드 예제에서는 강력하게 일관된 읽기를 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- DynamoDB 쿼리의 일관성 수준을 구성합니다.
- 강력하게 일관된 읽기를 사용하여 최신 데이터를 가져옵니다.
- 최종 일관성과 강력한 일관성의 장단점을 이해합니다.

SDK for JavaScript(v3)

를 사용하여 구성 가능한 읽기 일관성으로 DynamoDB 테이블을 쿼리합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */

```

```
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

PartiQL SELECT를 사용하여 데이터 쿼리

다음 코드 예제에서는 PartiQL SELECT 문을 사용하여 데이터를 쿼리하는 방법을 보여줍니다.

SDK for JavaScript(v3)

와 함께 PartiQL SELECT 문을 사용하여 DynamoDB 테이블에서 항목을 쿼리합니다 AWS SDK for JavaScript.

```
/**
 * This example demonstrates how to query items from a DynamoDB table using PartiQL.
 * It shows different ways to select data with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select an item by its primary key using PartiQL.
```

```

*
* @param tableName - The name of the DynamoDB table
* @param partitionKeyName - The name of the partition key attribute
* @param partitionKeyValue - The value of the partition key
* @returns The response from the ExecuteStatementCommand
*/
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
* Select an item by its composite key (partition key + sort key) using PartiQL.
*
* @param tableName - The name of the DynamoDB table
* @param partitionKeyName - The name of the partition key attribute
* @param partitionKeyValue - The value of the partition key
* @param sortKeyName - The name of the sort key attribute
* @param sortKeyValue - The value of the sort key
* @returns The response from the ExecuteStatementCommand
*/
export const selectItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,

```

```

    sortKeyValue: string | number
  ) => {
    const client = new DynamoDBClient({});
    const docClient = DynamoDBDocumentClient.from(client);

    const params = {
      Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
${sortKeyName} = ?`,
      Parameters: [partitionKeyValue, sortKeyValue],
    };

    try {
      const data = await docClient.send(new ExecuteStatementCommand(params));
      console.log("Item retrieved successfully");
      return data;
    } catch (err) {
      console.error("Error retrieving item:", err);
      throw err;
    }
  };
};

/**
 * Select items using a filter condition with PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));

```

```

        console.log("Items retrieved successfully");
        return data;
    } catch (err) {
        console.error("Error retrieving items:", err);
        throw err;
    }
};

/**
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix
 * @param prefix - The prefix to match
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByPrefix = async (
    tableName: string,
    attributeName: string,
    prefix: string
) => {
    const client = new DynamoDBClient({});
    const docClient = DynamoDBDocumentClient.from(client);

    const params = {
        Statement: `SELECT * FROM "${tableName}" WHERE
begins_with(${attributeName}, ?)`,
        Parameters: [prefix],
    };

    try {
        const data = await docClient.send(new ExecuteStatementCommand(params));
        console.log("Items retrieved successfully");
        return data;
    } catch (err) {
        console.error("Error retrieving items:", err);
        throw err;
    }
};

/**
 * Select items using a between condition for range queries.
 *

```

```

* @param tableName - The name of the DynamoDB table
* @param attributeName - The attribute to check for range
* @param startValue - The start value of the range
* @param endValue - The end value of the range
* @returns The response from the ExecuteStatementCommand
*/
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ? AND ?`,
    Parameters: [startValue, endValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");

  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456", "productId",
    "prod789");
};

```

```
// Select with a filter condition (can use any attribute)
await selectItemsWithFilter("UsersTable", "userType", "premium");

// Select items with a prefix (useful for hierarchical data)
await selectItemsByPrefix("ProductsTable", "category", "electronics");

// Select items within a range (useful for numeric or date ranges)
await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01", "2023-12-31");
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [BatchExecuteStatement](#)
 - [ExecuteStatement](#)

TTL 항목에 대한 쿼리

다음 코드 예제에서는 TTL 항목을 쿼리하는 방법을 보여줍니다.

SDK for JavaScript(v3)

필터링된 표현식을 쿼리하여를 사용하여 DynamoDB 테이블에서 TTL 항목을 수집합니다 AWS SDK for JavaScript.

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

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1') =>
{
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);

  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pk = :pk",
    FilterExpression: "#ea > :ea",
    ExpressionAttributeNames: {
```

```

        "#pk": "primaryKey",
        "#ea": "expireAt"
    },
    ExpressionAttributeValues: marshall({
        ":pk": primaryKey,
        ":ea": currentTime
    })
};

try {
    const { Items } = await client.send(new QueryCommand(params));
    Items.forEach(item => {
        console.log(unmarshall(item))
    });
    return Items;
} catch (err) {
    console.error(`Error querying items: ${err}`);
    throw err;
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

날짜 및 시간 패턴을 사용하여 테이블 쿼리

다음 코드 예제에서는 날짜 및 시간 패턴을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- DynamoDB에 날짜/시간 값을 저장하고 쿼리합니다.
- 정렬 키를 사용하여 날짜 범위 쿼리를 구현합니다.
- 효과적인 쿼리를 위해 날짜 문자열의 형식을 지정합니다.

SDK for JavaScript(v3)

를 사용하여 정렬 키의 날짜 범위를 사용하여 쿼리합니다 AWS SDK for JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");
```

```

/**
 * Queries a DynamoDB table for items within a specific date range on the sort key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
AND :endDate',
      ExpressionAttributeNames: {
        '#pk': partitionKeyName,
        '#sk': sortKeyName
      },
      ExpressionAttributeValues: {
        ':pkValue': { S: partitionKeyValue },
        ':startDate': { S: formattedStartDate },
        ':endDate': { S: formattedEndDate }
      }
    }
  }
}

```



```

    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}

```

에서 날짜-시간 변수를 사용하여 쿼리합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  dateKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();

```

```

    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
AND :endDate`,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#dateAttr": dateKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range: ${error}`);
    throw error;
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Query](#)를 참조하세요.

업데이트 표현식 순서 이해

다음 코드 예제에서는 업데이트 표현식 순서를 이해하는 방법을 보여줍니다.

- DynamoDB가 업데이트 표현식을 처리하는 방법을 알아봅니다.
- 업데이트 표현식의 연산 순서를 이해합니다.
- 표현식 평가를 이해하여 예상치 못한 결과를 방지합니다.

SDK for JavaScript(v3)

를 사용하여 업데이트 표현식 순서를 보여줍니다 AWS SDK for JavaScript.

```

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 * expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 * placeholders
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 * placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

```

```
};

// Add expression attribute names if provided
if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
}

// Add expression attribute values if provided
if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
}

// Execute the update
const response = await docClient.send(new UpdateCommand(updateParams));

return response;
}

/**
 * Demonstrate that variables hold copies of existing values before modifications.
 *
 * This function creates an item with initial values, then updates it with an
 * expression
 * that uses the values of attributes before they are modified in the same
 * expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 * demonstration
 */
async function demonstrateValueCopying(
    config,
    tableName,
    key
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Step 1: Create an item with initial values
    const initialItem = { ...key, a: 1, b: 2, c: 3 };
```

```
await docClient.send(new PutCommand({
  TableName: tableName,
  Item: initialItem
}));

// Step 2: Get the item to verify initial state
const responseBefore = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemBefore = responseBefore.Item || {};

// Step 3: Update the item with an expression that uses values before they are
modified
// This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to the
value of 'b'
const updateResponse = await docClient.send(new UpdateCommand({
  TableName: tableName,
  Key: key,
  UpdateExpression: "REMOVE a SET b = a, c = b",
  ReturnValues: "UPDATED_NEW"
}));

// Step 4: Get the item to verify final state
const responseAfter = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemAfter = responseAfter.Item || {};

// Return the results
return {
  initialState: itemBefore,
  updateResponse: updateResponse,
  finalState: itemAfter
};
}

/**
 * Demonstrate the order in which different action types are processed.
 */
```

```
* This function creates an item with initial values, then updates it with an
expression
* that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the order
* in which they are processed.
*
* @param {Object} config - AWS configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The primary key of the item to create and update
* @returns {Promise<Object>} - A dictionary containing the results of the
demonstration
*/
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
```

```

    const updateResponse = await docClient.send(new UpdateCommand({
      TableName: tableName,
      Key: key,
      UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
counter :increment DELETE set_attr :elements",
      ExpressionAttributeValues: {
        ":new_value": "Updated value",
        ":increment": 5,
        ":elements": new Set(["B"])
      },
      ReturnValues: "UPDATED_NEW"
    }));

    // Step 4: Get the item to verify final state
    const responseAfter = await docClient.send(new GetCommand({
      TableName: tableName,
      Key: key
    }));

    const itemAfter = responseAfter.Item || {};

    // Return the results
    return {
      initialState: itemBefore,
      updateResponse: updateResponse,
      finalState: itemAfter
    };
  }

  /**
   * Update multiple attributes with a single SET action.
   *
   * This function demonstrates how to update multiple attributes in a single SET
   * action,
   * which is more efficient than using multiple separate update operations.
   *
   * @param {Object} config - AWS configuration object
   * @param {string} tableName - The name of the DynamoDB table
   * @param {Object} key - The primary key of the item to update
   * @param {Object} attributes - The attributes to update and their new values
   * @returns {Promise<Object>} - The response from DynamoDB
   */
  async function updateWithMultipleSetActions(
    config,

```

```

    tableName,
    key,
    attributes
  ) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Build the update expression and expression attribute values
    let updateExpression = "SET ";
    const expressionAttributeValues = {};

    // Add each attribute to the update expression
    Object.entries(attributes).forEach(([attrName, attrValue], index) => {
      const valuePlaceholder = `:val${index}`;

      if (index > 0) {
        updateExpression += ", ";
      }
      updateExpression += `${attrName} = ${valuePlaceholder}`;

      expressionAttributeValues[valuePlaceholder] = attrValue;
    });

    // Execute the update
    const response = await docClient.send(new UpdateCommand({
      TableName: tableName,
      Key: key,
      UpdateExpression: updateExpression,
      ExpressionAttributeValues: expressionAttributeValues,
      ReturnValues: "UPDATED_NEW"
    }));

    return response;
  }

/**
 * Update an attribute with a value from another attribute or a default value.
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a value
 * from one attribute to another, or use a default value if the source doesn't
 * exist.
 *
 * @param {Object} config - AWS configuration object

```



```

* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The primary key of the item to update
* @param {string} sourceAttribute - The attribute to copy the value from
* @param {string} targetAttribute - The attribute to update
* @param {any} defaultValue - The default value to use if the source attribute
doesn't exist
* @returns {Promise<Object>} - The response from DynamoDB
*/
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Demonstrate complex update expressions with multiple operations on the same
attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object

```

```
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The primary key of the item to create and update
* @returns {Promise<Object>} - A dictionary containing the results of the
demonstration
*/
async function demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    list_attr: [1, 2, 3],
    map_attr: {
      nested1: "value1",
      nested2: "value2"
    }
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple operations on the same attributes
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `
      SET counter = counter + :inc1,
```

```

        counter = counter + :inc2,
        map_attr.nested1 = :new_val1,
        map_attr.nested3 = :new_val3,
        list_attr[0] = list_attr[1],
        list_attr[1] = list_attr[2]
    },
    ExpressionAttributeValues: {
        ":inc1": 5,
        ":inc2": 3,
        ":new_val1": "updated_value1",
        ":new_val3": "new_value3"
    },
    ReturnValues: "UPDATED_NEW"
}));

// Step 4: Get the item to verify final state
const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
}));

const itemAfter = responseAfter.Item || {};

// Return the results
return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
};
}

```

를 사용한 업데이트 표현식 순서 사용의 예입니다 AWS SDK for JavaScript.

```

/**
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
    // Example parameters
    const config = { region: "us-west-2" };
    const tableName = "OrderProcessing";

    console.log("Demonstrating update expression order of operations in DynamoDB");
}

```

```
try {
  // Example 1: Demonstrating value copying in update expressions
  console.log("\nExample 1: Demonstrating value copying in update expressions");
  const results1 = await demonstrateValueCopying(
    config,
    tableName,
    { OrderId: "order123" }
  );

  console.log("Initial state:", JSON.stringify(results1.initialState, null, 2));
  console.log("Update response:", JSON.stringify(results1.updateResponse, null,
2));
  console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

  console.log("\nExplanation:");
  console.log("1. The initial state had a=1, b=2, c=3");
  console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:");
  console.log("   - Copied the value of 'a' (which was 1) to be used for 'b'");
  console.log("   - Copied the value of 'b' (which was 2) to be used for 'c'");
  console.log("   - Removed the attribute 'a'");
  console.log("3. The final state has b=1, c=2, and 'a' is removed");
  console.log("4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications");

  // Example 2: Demonstrating the order of different action types
  console.log("\nExample 2: Demonstrating the order of different action types");
  const results2 = await demonstrateActionOrder(
    config,
    tableName,
    { OrderId: "order456" }
  );

  console.log("Initial state:", JSON.stringify(results2.initialState, null, 2));
  console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
  console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

  console.log("\nExplanation:");
  console.log("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE");
  console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
```

```
console.log("3. First, 'to_remove' was removed");
console.log("4. Then, 'to_modify' was set to a new value");
console.log("5. Next, 'counter' was incremented by 5");
console.log("6. Finally, 'B' was removed from the set attribute");

// Example 3: Updating multiple attributes in a single SET action
console.log("\nExample 3: Updating multiple attributes in a single SET action");
const response3 = await updateWithMultipleSetActions(
  config,
  tableName,
  { OrderId: "order789" },
  {
    Status: "Shipped",
    ShippingDate: "2025-05-28",
    TrackingNumber: "1Z999AA10123456784"
  }
);

console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

// Example 4: Conditional value copying with if_not_exists
console.log("\nExample 4: Conditional value copying with if_not_exists");
const response4 = await updateWithConditionalValueCopying(
  config,
  tableName,
  { OrderId: "order101" },
  "PreferredShippingMethod",
  "ShippingMethod",
  "Standard"
);

console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

// Example 5: Multiple operations on the same attribute
console.log("\nExample 5: Multiple operations on the same attribute");
const results5 = await demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  { OrderId: "order202" }
);

console.log("Initial state:", JSON.stringify(results5.initialState, null, 2));
```

```

    console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
    console.log("Final state:", JSON.stringify(results5.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The counter was incremented twice (first by 5, then by 3) for a
total of +8");
    console.log("2. The map attribute had one value updated and a new nested
attribute added");
    console.log("3. The list attribute had values shifted (value at index 1 moved to
index 0, value at index 2 moved to index 1)");
    console.log("4. All operations within the SET action are processed from left to
right");

    // Key points about update expression order of operations
    console.log("\nKey Points About Update Expression Order of Operations:");
    console.log("1. Variables in expressions hold copies of attribute values as they
existed BEFORE any modifications");
    console.log("2. Multiple actions in an update expression are processed in this
order: REMOVE, SET, ADD, DELETE");
    console.log("3. Within each action type, operations are processed from left to
right");
    console.log("4. You can reference the same attribute multiple times in an
expression");
    console.log("5. You can use if_not_exists() to conditionally set values based on
attribute existence");
    console.log("6. Using a single update expression with multiple actions is more
efficient than multiple separate updates");
    console.log("7. The update expression is atomic - either all actions succeed or
none do");

    } catch (error) {
        console.error("Error:", error);
    }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateItem](#)을 참조하세요.

테이블의 원 처리량 설정 업데이트

다음 코드 예제에서는 테이블의 원 처리량 설정을 업데이트하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript를 사용하여 기존 DynamoDB 테이블에서 읽 처리량 설정을 업데이트합니다.

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

export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
          Update: {
            IndexName: gsiName,
            WarmThroughput: {
              ReadUnitsPerSecond: gsiReadUnits,
              WriteUnitsPerSecond: gsiWriteUnits,
            },
          },
        },
      ],
      WarmThroughput: {
        ReadUnitsPerSecond: tableReadUnits,
        WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response: ${JSON.stringify(response)}`);
    return response;
  }
}
```

```

    } catch (error) {
      console.error(`Error updating table: ${error}`);
      throw error;
    }
  }

  // Example usage (commented out for testing)
  /*
  updateDynamoDBTableWarmThroughput(
    'example-table',
    5, 5,
    'example-index',
    2, 2
  );
  */

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateTable](#)을 참조하세요.

항목의 TTL 업데이트

다음 코드 예제에서는 항목의 TTL을 업데이트하는 방법을 보여줍니다.

SDK for JavaScript(v3)

```

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

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-east-1') => {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);
  const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

  const params = {
    TableName: tableName,
    Key: marshall({
      partitionKey: partitionKey,

```



```

        sortKey: sortKey
    })),
    UpdateExpression: "SET updatedAt = :c, expireAt = :e",
    ExpressionAttributeValues: marshall({
        ":c": currentTime,
        ":e": expireAt
    })),
};

try {
    const data = await client.send(new UpdateItemCommand(params));
    const responseData = unmarshall(data.Attributes);
    console.log("Item updated successfully: %s", responseData);
    return responseData;
} catch (err) {
    console.error("Error updating item:", err);
    throw err;
}
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-value');

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateItem](#)을 참조하세요.

PartiQL UPDATE를 사용하여 데이터 업데이트

다음 코드 예제에서는 PartiQL UPDATE 문을 사용하여 데이터를 업데이트하는 방법을 보여줍니다.

SDK for JavaScript(v3)

와 함께 PartiQL UPDATE 문을 사용하여 DynamoDB 테이블의 항목을 업데이트합니다 AWS SDK for JavaScript.

```

/**
 * This example demonstrates how to update items in a DynamoDB table using PartiQL.
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {

```

```

    DynamoDBDocumentClient,
    ExecuteStatementCommand,
    BatchExecuteStatementCommand,
  } from "@aws-sdk/lib-dynamodb";

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateSingleAttribute = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update multiple attributes of an item using PartiQL.
 *

```

```

* @param tableName - The name of the DynamoDB table
* @param partitionKeyName - The name of the partition key attribute
* @param partitionKeyValue - The value of the partition key
* @param attributeUpdates - Object containing attribute names and their new values
* @returns The response from the ExecuteStatementCommand
*/
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");

  // Create parameters array with attribute values followed by the partition key
  value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName} = ?`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item identified by a composite key (partition key + sort key) using
 * PartiQL.
 */

```

```

* @param tableName - The name of the DynamoDB table
* @param partitionKeyName - The name of the partition key attribute
* @param partitionKeyValue - The value of the partition key
* @param sortKeyName - The name of the sort key attribute
* @param sortKeyValue - The value of the sort key
* @param attributeName - The name of the attribute to update
* @param attributeValue - The new value for the attribute
* @returns The response from the ExecuteStatementCommand
*/
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item with a condition to ensure the update only happens if a condition
 * is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key

```

```

* @param attributeName - The name of the attribute to update
* @param attributeValue - The new value for the attribute
* @param conditionAttribute - The attribute to check in the condition
* @param conditionValue - The value to compare against in the condition
* @returns The response from the ExecuteStatementCommand
*/
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
* Batch update multiple items using PartiQL.
*
* @param tableName - The name of the DynamoDB table
* @param updates - Array of objects containing key and update information
* @returns The response from the BatchExecuteStatementCommand
*/
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{

```

```

    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each update
  const statements = updates.map((update) => {
    return {
      Statement: `UPDATE "${update.partitionKeyName}" SET ${update.attributeName} = ? WHERE
${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
  await updateSingleAttribute("UsersTable", "userId", "user123", "email",
    "newemail@example.com");

  // Update multiple attributes at once
  await updateMultipleAttributes("UsersTable", "userId", "user123", {
    email: "newemail@example.com",
    name: "John Smith",
  });
};

```

```
    lastLogin: new Date().toISOString(),
  });

// Update an item with a composite key (partition key + sort key)
await updateItemWithCompositeKey(
  "OrdersTable",
  "orderId",
  "order456",
  "productId",
  "prod789",
  "quantity",
  5
);

// Update with a condition
await updateItemWithCondition(
  "UsersTable",
  "userId",
  "user123",
  "userStatus",
  "active",
  "userType",
  "premium"
);

// Batch update multiple items
await batchUpdateItems("UsersTable", [
  {
    partitionKeyName: "userId",
    partitionKeyValue: "user123",
    attributeName: "lastLogin",
    attributeValue: new Date().toISOString(),
  },
  {
    partitionKeyName: "userId",
    partitionKeyValue: "user456",
    attributeName: "lastLogin",
    attributeValue: new Date().toISOString(),
  },
]);
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.

- [BatchExecuteStatement](#)
- [ExecuteStatement](#)

API Gateway를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon API Gateway에서 호출한 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

Lambda JavaScript 런타임 API를 사용하여 AWS Lambda 함수를 생성하는 방법을 보여줍니다. 이 예제에서는 다양한 AWS 서비스를 호출하여 특정 사용 사례를 수행합니다. 이 예제에서는 Amazon API Gateway에서 간접 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

원자성 카운터 작업 사용

다음 코드 예제에서는 DynamoDB에서 원자성 카운터 작업을 사용하는 방법을 보여줍니다.

- ADD 및 SET 작업을 사용하여 원자적으로 카운터를 증가시킵니다.
- 존재하지 않을 수 있는 카운터를 안전하게 증가시킵니다.
- 카운터 작업에 대한 낙관적 잠금을 구현합니다.

SDK for JavaScript(v3)

를 사용하여 원자성 카운터 작업을 시연합니다 AWS SDK for JavaScript.


```

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation

```

```

    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 * increments.
 * While this approach works, it's less idiomatic for simple increments than using
 * ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
    config,
    tableName,
    key,
    counterName,
    incrementValue
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using SET with an expression
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
        ExpressionAttributeValues: {
            ":increment": incrementValue
        },
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

```

```

    return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
    config,
    tableName,
    key,
    counterName,
    incrementValue,
    defaultValue = 0
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using SET with if_not_exists
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${counterName} = if_not_exists(${counterName}, :default)
+ :increment`,
        ExpressionAttributeValues: {
            ":increment": incrementValue,
            ":default": defaultValue
        },
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation

```

```

    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement optimistic
 * locking, which prevents race conditions when multiple processes try to update
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithLocking(
    config,
    tableName,
    key,
    counterName,
    incrementValue,
    expectedValue
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters with a condition expression
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
        ConditionExpression: `${counterName} = :expected`,
        ExpressionAttributeValues: {
            ":increment": incrementValue,
            ":expected": expectedValue
        },
        ReturnValues: "UPDATED_NEW"
    };
};

```

```

    try {
      // Perform the update operation
      const response = await docClient.send(new UpdateCommand(params));
      return {
        success: true,
        data: response
      };
    } catch (error) {
      // Check if the error is due to the condition check failing
      if (error.name === "ConditionalCheckFailedException") {
        return {
          success: false,
          error: "Optimistic locking failed: the counter value has changed"
        };
      }
      // Re-throw other errors
      throw error;
    }
  }
}

/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not found
 */
async function getCounterValue(
  config,
  tableName,
  key,
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {

```

```

    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
    : null;
}

```

를 사용한 원자성 카운터 작업 사용의 예입니다 AWS SDK for JavaScript.

```

/**
 * Example of how to use the atomic counter operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;

  console.log("Demonstrating different approaches to increment counters in
  DynamoDB");

  try {
    // Example 1: Using ADD operation (recommended for simple increments)
    console.log("\nExample 1: Incrementing counter with ADD operation");
    const response1 = await incrementCounterWithAdd(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);
  }
}

```

```
// Example 2: Using SET operation with an expression
console.log("\nExample 2: Incrementing counter with SET operation");
const response2 = await incrementCounterWithSet(
    config,
    tableName,
    key,
    counterName,
    incrementValue
);

console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

// Example 3: Safely incrementing a counter that might not exist
console.log("\nExample 3: Safely incrementing counter that might not exist");
const newKey = { ProductId: "P67890" };
const response3 = await incrementCounterSafely(
    config,
    tableName,
    newKey,
    counterName,
    incrementValue,
    0
);

console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

// Example 4: Incrementing with optimistic locking
console.log("\nExample 4: Incrementing with optimistic locking");

// First, get the current counter value
const currentValue = await getCounterValue(config, tableName, key, counterName);
console.log(`Current counter value: ${currentValue}`);

// Then, try to increment with optimistic locking
const response4 = await incrementCounterWithLocking(
    config,
    tableName,
    key,
    counterName,
    incrementValue,
    currentValue
);
```

```

    if (response4.success) {
        console.log(`Counter successfully incremented to:
${response4.data.Attributes[counterName]}`);
    } else {
        console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
    console.log("2. SET with expressions is more flexible for complex operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might not
exist");
    console.log("5. Optimistic locking can be added to either approach for
additional safety");

    } catch (error) {
        console.error("Error:", error);
    }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateItem](#)을 참조하세요.

조건부 작업 사용

다음 코드 예제에서는 DynamoDB에서 조건부 작업을 사용하는 방법을 보여줍니다.

- 데이터 덮어쓰기를 방지하기 위해 조건부 쓰기를 구현합니다.
- 조건 표현식을 사용하여 비즈니스 규칙을 적용합니다.
- 조건부 검사 실패를 원활하게 처리합니다.

SDK for JavaScript(v3)

를 사용하여 조건부 작업을 시연합니다 AWS SDK for JavaScript.

```

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
    DynamoDBDocumentClient,
    UpdateCommand,

```



```

    DeleteCommand,
    GetCommand,
    PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
    config,
    tableName,
    key,
    conditionAttribute,
    conditionValue,
    updateAttribute,
    updateValue
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters with a condition expression
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${updateAttribute} = :value`,
        ConditionExpression: `${conditionAttribute} = :condition`,
        ExpressionAttributeValues: {
            ":value": updateValue,
            ":condition": conditionValue
        },
        ReturnValues: "UPDATED_NEW"
    };
};

```

```

    try {
      // Perform the update operation
      const response = await docClient.send(new UpdateCommand(params));

      return {
        success: true,
        message: "Condition was met and update was performed",
        updatedAttributes: response.Attributes
      };
    } catch (error) {
      // Check if the error is due to the condition check failing
      if (error.name === "ConditionalCheckFailedException") {
        return {
          success: false,
          message: "Condition was not met, update was not performed",
          error: "ConditionalCheckFailedException"
        };
      }

      // Re-throw other errors
      throw error;
    }
  }

  /**
   * Perform a conditional delete operation.
   *
   * This function demonstrates how to delete an item only if a condition is met.
   *
   * @param {Object} config - AWS configuration object
   * @param {string} tableName - The name of the DynamoDB table
   * @param {Object} key - The key of the item to delete
   * @param {string} conditionAttribute - The attribute to check in the condition
   * @param {any} conditionValue - The value to compare against
   * @returns {Promise<Object>} - Result of the operation
   */
  async function conditionalDelete(
    config,
    tableName,
    key,
    conditionAttribute,
    conditionValue
  ) {

```

```
// Initialize the DynamoDB client
const client = new DynamoDBClient(config);
const docClient = DynamoDBDocumentClient.from(client);

// Define the delete parameters with a condition expression
const params = {
  TableName: tableName,
  Key: key,
  ConditionExpression: `${conditionAttribute} = :condition`,
  ExpressionAttributeValues: {
    ":condition": conditionValue
  },
  ReturnValues: "ALL_OLD"
};

try {
  // Perform the delete operation
  const response = await docClient.send(new DeleteCommand(params));

  return {
    success: true,
    message: "Condition was met and item was deleted",
    deletedItem: response.Attributes
  };
} catch (error) {
  // Check if the error is due to the condition check failing
  if (error.name === "ConditionalCheckFailedException") {
    return {
      success: false,
      message: "Condition was not met, item was not deleted",
      error: "ConditionalCheckFailedException"
    };
  }

  // Re-throw other errors
  throw error;
}

/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
 * to prevent race conditions when multiple processes update the same item.
 */
```

```

*
* @param {Object} config - AWS configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The key of the item to update
* @param {Object} updates - The attributes to update
* @param {number} expectedVersion - The expected current version number
* @returns {Promise<Object>} - Result of the operation
*/
async function updateWithOptimisticLocking(
  config,
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`val${index}`] = value;
  });

  // Add the version update
  updateExpressions.push("version = :newVersion");

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };
};

```

```

try {
  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return {
    success: true,
    message: "Update succeeded with optimistic locking",
    newVersion: expectedVersion + 1,
    updatedAttributes: response.Attributes
  };
} catch (error) {
  // Check if the error is due to the condition check failing
  if (error.name === "ConditionalCheckFailedException") {
    return {
      success: false,
      message: "Optimistic locking failed: the item was modified by another
process",
      error: "ConditionalCheckFailedException"
    };
  }

  // Re-throw other errors
  throw error;
}
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 * operation).
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
async function createIfNotExists(
  config,
  tableName,
  item
) {

```

```
// Initialize the DynamoDB client
const client = new DynamoDBClient(config);
const docClient = DynamoDBDocumentClient.from(client);

// Extract the primary key attributes
const keyAttributes = Object.keys(item).filter(attr =>
  attr === "id" || attr === "ID" || attr === "Id" ||
  attr.endsWith("Id") || attr.endsWith("ID") ||
  attr.endsWith("Key")
);

if (keyAttributes.length === 0) {
  throw new Error("Could not determine primary key attributes");
}

// Create a condition expression that checks if the item doesn't exist
const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

// Define the put parameters with a condition expression
const params = {
  TableName: tableName,
  Item: item,
  ConditionExpression: conditionExpression
};

try {
  // Perform the put operation
  await docClient.send(new PutCommand(params));

  return {
    success: true,
    message: "Item was created because it didn't exist",
    item
  };
} catch (error) {
  // Check if the error is due to the condition check failing
  if (error.name === "ConditionalCheckFailedException") {
    return {
      success: false,
      message: "Item already exists, creation was skipped",
      error: "ConditionalCheckFailedException"
    };
  }
}
```

```

        // Re-throw other errors
        throw error;
    }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
    config,
    tableName,
    key
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the get parameters
    const params = {
        TableName: tableName,
        Key: key
    };

    // Perform the get operation
    const response = await docClient.send(new GetCommand(params));

    // Return the item if it exists, otherwise null
    return response.Item || null;
}

```

에서 조건부 작업을 사용하는 예입니다 AWS SDK for JavaScript.

```

/**
 * Example of how to use conditional operations.
 */

```

```
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating conditional operations in DynamoDB");

  try {
    // Example 1: Conditional update based on attribute value
    console.log("\nExample 1: Conditional update based on attribute value");
    const updateResult = await conditionalUpdate(
      config,
      tableName,
      key,
      "Category",
      "Electronics",
      "Price",
      299.99
    );

    console.log(`Result: ${updateResult.message}`);
    if (updateResult.success) {
      console.log("Updated attributes:", updateResult.updatedAttributes);
    }

    // Example 2: Conditional delete based on attribute value
    console.log("\nExample 2: Conditional delete based on attribute value");
    const deleteResult = await conditionalDelete(
      config,
      tableName,
      key,
      "InStock",
      false
    );

    console.log(`Result: ${deleteResult.message}`);
    if (deleteResult.success) {
      console.log("Deleted item:", deleteResult.deletedItem);
    }

    // Example 3: Optimistic locking with version number
    console.log("\nExample 3: Optimistic locking with version number");
  }
}
```



```
// First, get the current item to check its version
const currentItem = await getItem(config, tableName, { ProductId: "P67890" });
const currentVersion = currentItem ? (currentItem.version || 0) : 0;

console.log(`Current version: ${currentVersion}`);

// Then, update with optimistic locking
const lockingResult = await updateWithOptimisticLocking(
  config,
  tableName,
  { ProductId: "P67890" },
  {
    Name: "Updated Product Name",
    Description: "This is an updated description"
  },
  currentVersion
);

console.log(`Result: ${lockingResult.message}`);
if (lockingResult.success) {
  console.log(`New version: ${lockingResult.newVersion}`);
  console.log("Updated attributes:", lockingResult.updatedAttributes);
}

// Example 4: Create item only if it doesn't exist
console.log("\nExample 4: Create item only if it doesn't exist");
const createResult = await createIfNotExists(
  config,
  tableName,
  {
    ProductId: "P99999",
    Name: "New Product",
    Category: "Accessories",
    Price: 19.99,
    InStock: true
  }
);

console.log(`Result: ${createResult.message}`);
if (createResult.success) {
  console.log("Created item:", createResult.item);
}

// Explain conditional operations
```

```

    console.log("\nKey points about conditional operations:");
    console.log("1. Conditional operations only succeed if the condition is met");
    console.log("2. ConditionalCheckFailedException indicates the condition wasn't met");
    console.log("3. Optimistic locking prevents race conditions in concurrent updates");
    console.log("4. attribute_exists and attribute_not_exists are useful for checking if attributes are present");
    console.log("5. Conditional operations are atomic - they either succeed completely or fail completely");
    console.log("6. You can use any valid comparison operators and functions in condition expressions");
    console.log("7. Conditional operations don't consume write capacity if the condition fails");

    } catch (error) {
        console.error("Error:", error);
    }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.

- [DeleteItem](#)
- [PutItem](#)
- [UpdateItem](#)

표현식 속성 이름 사용

다음 코드 예제에서는 DynamoDB에서 표현식 속성 이름을 사용하는 방법을 보여줍니다.

- DynamoDB 표현식에서 예약어로 작업합니다.
- 표현식 속성 이름 자리 표시자를 사용합니다.
- 속성 이름의 특수 문자를 처리합니다.

SDK for JavaScript(v3)

를 사용하여 표현식 속성 이름을 보여줍니다 AWS SDK for JavaScript.

```

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {

```

```

    DynamoDBDocumentClient,
    UpdateCommand,
    GetCommand,
    QueryCommand,
    ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateReservedWordAttribute(
    config,
    tableName,
    key,
    reservedWordAttribute,
    value
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using expression attribute names
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: "SET #attr = :value",
        ExpressionAttributeNames: {
            "#attr": reservedWordAttribute
        },
        ExpressionAttributeValues: {
            ":value": value
        },
        ReturnValues: "UPDATED_NEW"
    };
};

```

```

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters to
 * update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
    config,
    tableName,
    key,
    specialCharAttribute,
    value
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using expression attribute names
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: "SET #attr = :value",
        ExpressionAttributeNames: {
            "#attr": specialCharAttribute
        },
        ExpressionAttributeValues: {
            ":value": value
        },
        ReturnValues: "UPDATED_NEW"
    };

```

```

};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
 * @param {string} reservedWordAttribute - The reserved word attribute to filter on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the query parameters using expression attribute names
  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pkName = :pkValue",
    FilterExpression: "#attr = :value",
    ExpressionAttributeNames: {
      "#pkName": partitionKeyName,
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {

```

```

        ":pkValue": partitionKeyValue,
        ":value": value
    }
};

// Perform the query operation
const response = await docClient.send(new QueryCommand(params));

return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedReservedWordAttribute(
    config,
    tableName,
    key,
    attributePath,
    value
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Create expression attribute names for each part of the path
    const expressionAttributeNames = {};
    for (let i = 0; i < attributePath.length; i++) {
        expressionAttributeNames[`#attr${i}`] = attributePath[i];
    }

    // Build the attribute path using the expression attribute names
    const attributePathExpression = attributePath
        .map((_, i) => `#attr${i}`)

```

```

        .join(".");

// Define the update parameters
const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: {
        ":value": value
    },
    ReturnValues: "UPDATED_NEW"
};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
 * in a complex filter expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function scanWithMultipleAttributeNames(
    config,
    tableName,
    filters
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Create expression attribute names and values
    const expressionAttributeNames = {};
    const expressionAttributeValues = {};
    const filterConditions = [];

```

```

// Build the filter expression
Object.entries(filters).forEach(([attrName, value], index) => {
  const nameKey = `#attr${index}`;
  const valueKey = `:val${index}`;

  expressionAttributeNames[nameKey] = attrName;
  expressionAttributeValues[valueKey] = value;
  filterConditions.push(`${nameKey} = ${valueKey}`);
});

// Join the filter conditions with AND
const filterExpression = filterConditions.join(" AND ");

// Define the scan parameters
const params = {
  TableName: tableName,
  FilterExpression: filterExpression,
  ExpressionAttributeNames: expressionAttributeNames,
  ExpressionAttributeValues: expressionAttributeValues
};

// Perform the scan operation
const response = await docClient.send(new ScanCommand(params));

return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client

```



```

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

// Define the get parameters
const params = {
  TableName: tableName,
  Key: key
};

// Perform the get operation
const response = await docClient.send(new GetCommand(params));

// Return the item if it exists, otherwise null
return response.Item || null;
}

```

에서 표현식 속성 이름의 사용 예제입니다 AWS SDK for JavaScript.

```

/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating expression attribute names in DynamoDB");

  try {
    // Example 1: Update an attribute that is a reserved word
    console.log("\nExample 1: Updating an attribute that is a reserved word");
    const response1 = await updateReservedWordAttribute(
      config,
      tableName,
      key,
      "Size", // "SIZE" is a reserved word in DynamoDB
      "Large"
    );

    console.log("Updated attribute:", response1.Attributes);
  }
}

```

```
// Example 2: Update an attribute with special characters
console.log("\nExample 2: Updating an attribute with special characters");
const response2 = await updateSpecialCharacterAttribute(
    config,
    tableName,
    key,
    "Product-Type", // Contains a hyphen, which is a special character
    "Electronics"
);

console.log("Updated attribute:", response2.Attributes);

// Example 3: Query with a reserved word attribute
console.log("\nExample 3: Querying with a reserved word attribute");
const response3 = await queryWithReservedWordAttribute(
    config,
    tableName,
    "Category",
    "Electronics",
    "Count", // "COUNT" is a reserved word in DynamoDB
    10
);

console.log(`Found ${response3.Items.length} items`);

// Example 4: Update a nested attribute with reserved words in the path
console.log("\nExample 4: Updating a nested attribute with reserved words in the path");
const response4 = await updateNestedReservedWordAttribute(
    config,
    tableName,
    key,
    ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
    30
);

console.log("Updated nested attribute:", response4.Attributes);

// Example 5: Scan with multiple attribute name placeholders
console.log("\nExample 5: Scanning with multiple attribute name placeholders");
const response5 = await scanWithMultipleAttributeNames(
    config,
    tableName,
    {
```

```
        "Size": "Large",
        "Count": 10,
        "Product-Type": "Electronics"
    }
);

console.log(`Found ${response5.Items.length} items`);

// Get the final state of the item
console.log("\nFinal state of the item:");
const item = await getItem(config, tableName, key);
console.log(JSON.stringify(item, null, 2));

// Show some common reserved words
console.log("\nSome common DynamoDB reserved words:");
const commonReservedWords = [
    "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
    "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
    "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
    "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
    "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN", "NAME",
    "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
    "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
    "VIEW", "WHERE"
];
console.log(commonReservedWords.join(", "));

// Explain expression attribute names
console.log("\nKey points about expression attribute names:");
console.log("1. Use expression attribute names (#name) for reserved words");
console.log("2. Use expression attribute names for attributes with special characters");
console.log("3. Special characters include: spaces, hyphens, dots, and other non-alphanumeric characters");
console.log("4. Expression attribute names are required for nested attributes with reserved words");
console.log("5. You can use multiple expression attribute names in a single expression");
console.log("6. Expression attribute names are case-sensitive");
console.log("7. Expression attribute names are only used in expressions, not in the actual data");

} catch (error) {
    console.error("Error:", error);
}
```

```
}  
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [Query](#)
 - [UpdateItem](#)

예약된 이벤트를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon EventBridge 예약 이벤트에서 호출된 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS Lambda 함수를 호출하는 Amazon EventBridge 예약 이벤트를 생성하는 방법을 보여줍니다. Lambda 함수가 간접 호출될 때 cron 표현식을 사용하여 일정을 예약하도록 EventBridge를 구성합니다. 이 예제에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

서버리스 예제

DynamoDB 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제에서는 DynamoDB 스트림에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DynamoDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
};

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

TypeScript를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
export const handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
}
```

```
const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

DynamoDB 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제에서는 DynamoDB 스트림에서 이벤트를 수신하는 Lambda 함수에 대한 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

TypeScript를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

SDK for JavaScript (v3)를 사용한 Amazon EC2 예

다음 코드 예제에서는 Amazon EC2에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)
- [시나리오](#)

시작하기

Hello Amazon EC2

다음 코드 예제에서는 Amazon EC2 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  const client = new EC2Client();
  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);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeSecurityGroups](#)를 참조하세요.

기본 사항

기본 사항 알아보기

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.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 파일에는 EC2에서 자주 사용되는 작업 목록이 포함되어 있습니다. 단계는 대화형 예제 실행을 간소화하는 시나리오 프레임워크로 구성됩니다. 전체 컨텍스트는 GitHub 리포지토리를 참조하세요.

```

import { tmpdir } from "node:os";
import { writeFile, mkdtemp, rm } from "node:fs/promises";

```

```

import { join } from "node:path";
import { get } from "node:http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
  DeleteSecurityGroupCommand,
  DisassociateAddressCommand,
  paginateDescribeImages,
  paginateDescribeInstances,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

/**
 * @typedef {{
 *   ec2Client: import('@aws-sdk/client-ec2').EC2Client,
 *   errors: Error[],
 *   keyPairId?: string,
 *   tmpDirectory?: string,
 *   securityGroupId?: string,
 *   ipAddress?: string,
 *   images?: import('@aws-sdk/client-ec2').Image[],
 *   image?: import('@aws-sdk/client-ec2').Image,
 *   instanceTypes?: import('@aws-sdk/client-ec2').InstanceTypeInfo[],

```

```

*   instanceId?: string,
*   instanceIpAddress?: string,
*   allocationId?: string,
*   allocatedIpAddress?: string,
*   associationId?: string,
* }} State
*/

/**
 * A skip function provided to the `skipWhen` of a Step when you want
 * to ignore that step if any errors have occurred.
 * @param {State} state
 */
const skipWhenErrors = (state) => state.errors.length > 0;

const MAX_WAITER_TIME_IN_SECONDS = 60 * 8;

export const confirm = new ScenarioInput("confirmContinue", "Continue?", {
  type: "confirm",
  skipWhen: skipWhenErrors,
});

export const exitOnNoConfirm = new ScenarioAction(
  "exitOnConfirmContinueFalse",
  (/** @type { { earlyExit: boolean } & Record<string, any> } */ state) => {
    if (!state[confirm.name]) {
      state.earlyExit = true;
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

export const greeting = new ScenarioOutput(
  "greeting",
  `
Welcome to the Amazon EC2 basic usage scenario.

Before you launch an instances, you'll need to provide a few things:
- A key pair - This is for SSH access to your EC2 instance. You only need to
provide the name.

```

```

- A security group - This is used for configuring access to your instance. Again,
only the name is needed.
- An IP address - Your public IP address will be fetched.
- An Amazon Machine Image (AMI)
- A compatible instance type`,
  { header: true, preformatted: true, skipWhen: skipWhenErrors },
);

export const provideKeyPairName = new ScenarioInput(
  "keyPairName",
  "Provide a name for a new key pair.",
  { type: "input", default: "ec2-example-key-pair", skipWhen: skipWhenErrors },
);

export const createKeyPair = new ScenarioAction(
  "createKeyPair",
  async (** @type {State} */ state) => {
    try {
      // Create a key pair in Amazon EC2.
      const { KeyMaterial, KeyPairId } = await state.ec2Client.send(
        // A unique name for the key pair. Up to 255 ASCII characters.
        new CreateKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
      );

      state.keyPairId = KeyPairId;

      // Save the private key in a temporary location.
      state.tmpDirectory = await mkdtemp(join(tmpdir(), "ec2-scenario-tmp"));
      await writeFile(
        `${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem`,
        KeyMaterial,
        {
          mode: 0o400,
        },
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidKeyPair.Duplicate"
      ) {
        caught.message = `${caught.message}. Try another key name.`;
      }

      state.errors.push(caught);
    }
  }
);

```

```

    }
  },
  { skipWhen: skipWhenErrors },
);

export const logKeyPair = new ScenarioOutput(
  "logKeyPair",
  (** @type {State} */ state) =>
    `Created the key pair ${state[provideKeyPairName.name]}.`,
  { skipWhen: skipWhenErrors },
);

export const confirmDeleteKeyPair = new ScenarioInput(
  "confirmDeleteKeyPair",
  "Do you want to delete the key pair?",
  {
    type: "confirm",
    // Don't do anything when a key pair was never created.
    skipWhen: (** @type {State} */ state) => !state.keyPairId,
  },
);

export const maybeDeleteKeyPair = new ScenarioAction(
  "deleteKeyPair",
  async (** @type {State} */ state) => {
    try {
      // Delete a key pair by name from EC2
      await state.ec2Client.send(
        new DeleteKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        // Occurs when a required parameter (e.g. KeyName) is undefined.
        caught.name === "MissingParameter"
      ) {
        caught.message = `${caught.message}. Did you provide the required value?`;
      }
      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no key pair to delete or the user chooses
    // to keep it.
  }
);

```

```

    skipWhen: (/** @type {State} */ state) =>
      !state.keyPairId || !state[confirmDeleteKeyPair.name],
  },
);

export const provideSecurityGroupName = new ScenarioInput(
  "securityGroupName",
  "Provide a name for a new security group.",
  { type: "input", default: "ec2-scenario-sg", skipWhen: skipWhenErrors },
);

export const createSecurityGroup = new ScenarioAction(
  "createSecurityGroup",
  async (/** @type {State} */ state) => {
    try {
      // Create a new security group that will be used to configure ingress/egress
      for
      // an EC2 instance.
      const { GroupId } = await state.ec2Client.send(
        new CreateSecurityGroupCommand({
          GroupName: state[provideSecurityGroupName.name],
          Description: "A security group for the Amazon EC2 example.",
        }),
      );
      state.securityGroupId = GroupId;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidGroup.Duplicate") {
        caught.message = `${caught.message}. Please provide a different name for
your security group.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSecurityGroup = new ScenarioOutput(
  "logSecurityGroup",
  (/** @type {State} */ state) =>
    `Created the security group ${state.securityGroupId}.`,
  { skipWhen: skipWhenErrors },
);

```

```

export const confirmDeleteSecurityGroup = new ScenarioInput(
  "confirmDeleteSecurityGroup",
  "Do you want to delete the security group?",
  {
    type: "confirm",
    // Don't do anything when a security group was never created.
    skipWhen: (/** @type {State} */ state) => !state.securityGroupId,
  },
);

export const maybeDeleteSecurityGroup = new ScenarioAction(
  "deleteSecurityGroup",
  async (/** @type {State} */ state) => {
    try {
      // Delete the security group if the 'skipWhen' condition below is not met.
      await state.ec2Client.send(
        new DeleteSecurityGroupCommand({
          GroupId: state.securityGroupId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidGroupId.Malformed"
      ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
      }
      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no security group to delete
    // or the user chooses to keep it.
    skipWhen: (/** @type {State} */ state) =>
      !state.securityGroupId || !state[confirmDeleteSecurityGroup.name],
  },
);

export const authorizeSecurityGroupIngress = new ScenarioAction(
  "authorizeSecurity",
  async (/** @type {State} */ state) => {
    try {
      // Get the public IP address of the machine running this example.
      const ipAddress = await 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);
    });
  });
  state.ipAddress = ipAddress;
  // Allow ingress from the IP address above to the security group.
  // This will allow you to SSH into the EC2 instance.
  const command = new AuthorizeSecurityGroupIngressCommand({
    GroupId: state.securityGroupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  await state.ec2Client.send(command);
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "InvalidGroupId.Malformed"
  ) {
    caught.message = `${caught.message}. Please provide a valid GroupId.`;
  }

  state.errors.push(caught);
}
},
{ skipWhen: skipWhenErrors },
);

export const logSecurityGroupIngress = new ScenarioOutput(
  "logSecurityGroupIngress",
  (** @type {State} */ state) =>
    `Allowed SSH access from your public IP: ${state.ipAddress}.`,

```



```

    { skipWhen: skipWhenErrors },
  );

export const getImages = new ScenarioAction(
  "images",
  async (** @type {State} */ state) => {
    const AMIs = [];
    // Some AWS services publish information about common artifacts as AWS Systems
    Manager (SSM)
    // public parameters. For example, the Amazon Elastic Compute Cloud (Amazon EC2)
    // service publishes information about Amazon Machine Images (AMIs) as public
    parameters.

    // Create the paginator for getting images. Actions that return multiple pages
    of
    // results have paginators to simplify those calls.
    const getParametersByPathPaginator = paginateGetParametersByPath(
      {
        // Not storing this client in state since it's only used once.
        client: new SSMClient({}),
      },
      {
        // The path to the public list of the latest amazon-linux instances.
        Path: "/aws/service/ami-amazon-linux-latest",
      },
    );

    try {
      for await (const page of getParametersByPathPaginator) {
        for (const param of page.Parameters) {
          // Filter by Amazon Linux 2
          if (param.Name.includes("amzn2")) {
            AMIs.push(param.Value);
          }
        }
      }
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidFilterValue") {
        caught.message = `${caught.message} Please provide a valid filter value for
        paginateGetParametersByPath.`;
      }
      state.errors.push(caught);
      return;
    }
  }
);

```

```

const imageDetails = [];
const describeImagesPaginator = paginateDescribeImages(
  { client: state.ec2Client },
  // The images found from the call to SSM.
  { ImageIds: AMIs },
);

try {
  // Get more details for the images found above.
  for await (const page of describeImagesPaginator) {
    imageDetails.push(...(page.Images || []));
  }

  // Store the image details for later use.
  state.images = imageDetails;
} catch (caught) {
  if (caught instanceof Error && caught.name === "InvalidAMIID.NotFound") {
    caught.message = `${caught.message}. Please provide a valid image id.`;
  }

  state.errors.push(caught);
},
{ skipWhen: skipWhenErrors },
);

export const provideImage = new ScenarioInput(
  "image",
  "Select one of the following images.",
  {
    type: "select",
    choices: (/** @type { State } */ state) =>
      state.images.map((image) => ({
        name: `${image.Description}`,
        value: image,
      })),
    default: (/** @type { State } */ state) => state.images[0],
    skipWhen: skipWhenErrors,
  },
);

export const getCompatibleInstanceTypes = new ScenarioAction(
  "getCompatibleInstanceTypes",

```

```

async (** @type {State} */ state) => {
  // Get more details about instance types that match the architecture of
  // the provided image.
  const paginator = paginateDescribeInstanceTypes(
    { client: state.ec2Client, pageSize: 25 },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          // The value selected from provideImage()
          Values: [state.image.Architecture],
        },
        // Filter for smaller, less expensive, types.
        { Name: "instance-type", Values: ["*.micro", "/*.small"] },
      ],
    },
  );

  const instanceTypes = [];

  try {
    for await (const page of paginator) {
      if (page.InstanceTypes.length) {
        instanceTypes.push(...(page.InstanceTypes || []));
      }
    }

    if (!instanceTypes.length) {
      state.errors.push(
        "No instance types matched the instance type filters.",
      );
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      caught.message = `${caught.message}. Please check the provided values and
try again.`;
    }

    state.errors.push(caught);
  }

  state.instanceTypes = instanceTypes;
},
{ skipWhen: skipWhenErrors },

```

```

);

export const provideInstanceType = new ScenarioInput(
  "instanceType",
  "Select an instance type.",
  {
    choices: (/** @type {State} */ state) =>
      state.instanceTypes.map((instanceType) => ({
        name: `${instanceType.InstanceType} - Memory:
${instanceType.MemoryInfo.SizeInMiB}`,
        value: instanceType.InstanceType,
      })),
    type: "select",
    default: (/** @type {State} */ state) =>
      state.instanceTypes[0].InstanceType,
    skipWhen: skipWhenErrors,
  },
);

export const runInstance = new ScenarioAction(
  "runInstance",
  async (/** @type { State } */ state) => {
    const { Instances } = await state.ec2Client.send(
      new RunInstancesCommand({
        KeyName: state[provideKeyPairName.name],
        SecurityGroupIds: [state.securityGroupId],
        ImageId: state.image.ImageId,
        InstanceType: state[provideInstanceType.name],
        // Availability Zones have capacity limitations that may impact your ability
        // to launch instances.
        // The `RunInstances` operation will only succeed if it can allocate at
        // least the `MinCount` of instances.
        // However, EC2 will attempt to launch up to the `MaxCount` of instances,
        // even if the full request cannot be satisfied.
        // If you need a specific number of instances, use `MinCount` and `MaxCount`
        // set to the same value.
        // If you want to launch up to a certain number of instances, use `MaxCount`
        // and let EC2 provision as many as possible.
        // If you require a minimum number of instances, but do not want to exceed a
        // maximum, use both `MinCount` and `MaxCount`.
        MinCount: 1,
        MaxCount: 1,
      })),
  },
);

```

```

state.instanceId = Instances[0].InstanceId;

try {
  // Poll `DescribeInstanceStatus` until status is "ok".
  await waitUntilInstanceStatusOk(
    {
      client: state.ec2Client,
      maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
    },
    { InstanceIds: [Instances[0].InstanceId] },
  );
} catch (caught) {
  if (caught instanceof Error && caught.name === "TimeoutError") {
    caught.message = `${caught.message}. Try increasing the maxWaitTime in the
waiter.`;
  }

  state.errors.push(caught);
}
},
{ skipWhen: skipWhenErrors },
);

export const logRunInstance = new ScenarioOutput(
  "logRunInstance",
  "The next step is to run your EC2 instance for the first time. This can take a few
minutes.",
  { header: true, skipWhen: skipWhenErrors },
);

export const describeInstance = new ScenarioAction(
  "describeInstance",
  async (/** @type { State } */ state) => {
    /** @type { import("@aws-sdk/client-ec2").Instance[] } */
    const instances = [];

    try {
      const paginator = paginateDescribeInstances(
        {
          client: state.ec2Client,
        },
        {
          // Only get our created instance.

```

```

        InstanceIds: [state.instanceId],
      },
    );

    for await (const page of paginator) {
      for (const reservation of page.Reservations) {
        instances.push(...reservation.InstanceIds);
      }
    }
    if (instances.length !== 1) {
      throw new Error(`Instance ${state.instanceId} not found.`);
    }

    // The only info we need is the IP address for SSH purposes.
    state.instanceIpAddress = instances[0].PublicIpAddress;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      caught.message = `${caught.message}. Please check provided values and try
again.`;
    }

    state.errors.push(caught);
  }
},
{ skipWhen: skipWhenErrors },
);

export const logSSHConnectionInfo = new ScenarioOutput(
  "logSSHConnectionInfo",
  (/** @type { State } */ state) =>
    `You can now SSH into your instance using the following command:
ssh -i ${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem ec2-user@
${state.instanceIpAddress}`,
  { preformatted: true, skipWhen: skipWhenErrors },
);

export const logStopInstance = new ScenarioOutput(
  "logStopInstance",
  "Stopping your EC2 instance.",
  { skipWhen: skipWhenErrors },
);

export const stopInstance = new ScenarioAction(
  "stopInstance",

```

```

async (/** @type { State } */ state) => {
  try {
    await state.ec2Client.send(
      new StopInstancesCommand({
        InstanceIds: [state.instanceId],
      }),
    );

    await waitUntilInstanceStopped(
      {
        client: state.ec2Client,
        maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
      },
      { InstanceIds: [state.instanceId] },
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "TimeoutError") {
      caught.message = `${caught.message}. Try increasing the maxWaitTime in the
waiter.`;
    }

    state.errors.push(caught);
  }
},
// Don't try to stop an instance that doesn't exist.
{ skipWhen: (/** @type { State } */ state) => !state.instanceId },
);

export const logIpAddressBehavior = new ScenarioOutput(
  "logIpAddressBehavior",
  [
    "When you run an instance, by default it's assigned an IP address.",
    "That IP address is not static. It will change every time the instance is
restarted.",
    "The next step is to stop and restart your instance to demonstrate this
behavior.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const logStartInstance = new ScenarioOutput(
  "logStartInstance",
  (/** @type { State } */ state) => `Starting instance ${state.instanceId}`,
  { skipWhen: skipWhenErrors },
);

```

```

);

export const startInstance = new ScenarioAction(
  "startInstance",
  async (** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StartInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStatusOk(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
        waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logIpAllocation = new ScenarioOutput(
  "logIpAllocation",
  [
    "It is possible to have a static IP address.",
    "To demonstrate this, an IP will be allocated and associated to your EC2
    instance.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const allocateIp = new ScenarioAction(
  "allocateIp",
  async (** @type { State } */ state) => {

```



```

    try {
      // An Elastic IP address is allocated to your AWS account, and is yours until
      you release it.
      const { AllocationId, PublicIp } = await state.ec2Client.send(
        new AllocateAddressCommand({}),
      );
      state.allocationId = AllocationId;
      state.allocatedIpAddress = PublicIp;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        caught.message = `${caught.message}. Did you provide these values?`;
      }
      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const associateIp = new ScenarioAction(
  "associateIp",
  async (/** @type { State } */ state) => {
    try {
      // Associate an allocated IP address to an EC2 instance. An IP address can be
      allocated
      // with the AllocateAddress action.
      const { AssociationId } = await state.ec2Client.send(
        new AssociateAddressCommand({
          AllocationId: state.allocationId,
          InstanceId: state.instanceId,
        }),
      );
      state.associationId = AssociationId;
      // Update the IP address that is being tracked to match
      // the one just associated.
      state.instanceIpAddress = state.allocatedIpAddress;
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Did you provide the ID of a valid
        Elastic IP address AllocationId?`;
      }
      state.errors.push(caught);
    }
  }
);

```

```
    }  
  },  
  { skipWhen: skipWhenErrors },  
);  
  
export const logStaticIpProof = new ScenarioOutput(  
  "logStaticIpProof",  
  "The IP address should remain the same even after stopping and starting the  
  instance.",  
  { header: true, skipWhen: skipWhenErrors },  
);  
  
export const logCleanUp = new ScenarioOutput(  
  "logCleanUp",  
  "That's it! You can choose to clean up the resources now, or clean them up on your  
  own later.",  
  { header: true, skipWhen: skipWhenErrors },  
);  
  
export const confirmDisassociateAddress = new ScenarioInput(  
  "confirmDisassociateAddress",  
  "Do you want to disassociate and release the static IP address created earlier?",  
  {  
    type: "confirm",  
    skipWhen: (/** @type { State } */ state) => !state.associationId,  
  },  
);  
  
export const maybeDisassociateAddress = new ScenarioAction(  
  "maybeDisassociateAddress",  
  async (/** @type { State } */ state) => {  
    try {  
      await state.ec2Client.send(  
        new DisassociateAddressCommand({  
          AssociationId: state.associationId,  
        })),  
    );  
  } catch (caught) {  
    if (  
      caught instanceof Error &&  
      caught.name === "InvalidAssociationID.NotFound"  
    ) {  
      caught.message = `${caught.message}. Please provide a valid association  
      ID.`;  
    }  
  }  
}
```

```

    }
    state.errors.push(caught);
  }
},
{
  skipWhen: (/** @type { State } */ state) =>
    !state[confirmDisassociateAddress.name] || !state.associationId,
},
);

export const maybeReleaseAddress = new ScenarioAction(
  "maybeReleaseAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new ReleaseAddressCommand({
          AllocationId: state.allocationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid AllocationID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.allocationId,
  },
);

export const confirmTerminateInstance = new ScenarioInput(
  "confirmTerminateInstance",
  "Do you want to terminate the instance?",
  // Don't do anything when an instance was never run.
  {
    skipWhen: (/** @type { State } */ state) => !state.instanceId,
    type: "confirm",
  },
);

```

```

export const maybeTerminateInstance = new ScenarioAction(
  "terminateInstance",
  async (** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new TerminateInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );
      await waitUntilInstanceTerminated(
        { client: state.ec2Client },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
        waiter.`;
      }

      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no instance to terminate or the
    // user chooses not to terminate.
    skipWhen: (** @type { State } */ state) =>
      !state.instanceId || !state[confirmTerminateInstance.name],
  },
);

export const deleteTemporaryDirectory = new ScenarioAction(
  "deleteTemporaryDirectory",
  async (** @type { State } */ state) => {
    try {
      await rm(state.tmpDirectory, { recursive: true });
    } catch (caught) {
      state.errors.push(caught);
    }
  },
);

export const logErrors = new ScenarioOutput(
  "logErrors",

```

```

    (/** @type {State} */ state) => {
      const errorList = state.errors
        .map((err) => ` - ${err.name}: ${err.message}`)
        .join("\n");
      return `Scenario errors found:\n${errorList}`;
    },
    {
      preformatted: true,
      header: true,
      // Don't log errors when there aren't any!
      skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
    },
  );

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 항목을 참조하세요.

- [AllocateAddress](#)
- [AssociateAddress](#)
- [AuthorizeSecurityGroupIngress](#)
- [CreateKeyPair](#)
- [CreateSecurityGroup](#)
- [DeleteKeyPair](#)
- [DeleteSecurityGroup](#)
- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)

작업

AllocateAddress

다음 코드 예시는 AllocateAddress의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { AllocateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Allocates an Elastic IP address to your AWS account.
 */
export const main = async () => {
  const client = new EC2Client({});
  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: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2. Look under Network & Security > Elastic IPs",
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};

import { fileURLToPath } from "node:url";
// Call function if run directly.
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [AllocateAddress](#)를 참조하세요.

AssociateAddress

다음 코드 예시는 AssociateAddress의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { AssociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Associates an Elastic IP address, or carrier IP address (for instances that are
 * in subnets in Wavelength Zones)
 * with an instance or a network interface.
 * @param {{ instanceId: string, allocationId: string }} options
 */
export const main = async ({ instanceId, allocationId }) => {
  const client = new EC2Client({});
  const command = new AssociateAddressCommand({
    // You need to allocate an Elastic IP address before associating it with an
    // instance.
    // You can do that with the AllocateAddressCommand.
    AllocationId: allocationId,
    // You need to create an EC2 instance before an IP address can be associated
    // with it.
    // You can do that with the RunInstancesCommand.
    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 ${AssociationId}.`,
    );
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Did you provide the ID of a valid Elastic IP address
AllocationId?`,
      );
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [AssociateAddress](#)를 참조하세요.

AuthorizeSecurityGroupIngress

다음 코드 예시는 AuthorizeSecurityGroupIngress의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import {
  AuthorizeSecurityGroupIngressCommand,
  EC2Client,
} from "@aws-sdk/client-ec2";

/**

```



```

* Adds the specified inbound (ingress) rules to a security group.
* @param {{ groupId: string, ipAddress: string }} options
*/
export const main = async ({ groupId, ipAddress }) => {
  const client = new EC2Client({});
  const command = new AuthorizeSecurityGroupIngressCommand({
    // Use a group ID from the AWS console or
    // the DescribeSecurityGroupsCommand.
    GroupId: groupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        // 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: `${ipAddress}/32` }],
      },
    ],
  });

  try {
    const { SecurityGroupRules } = await client.send(command);
    console.log(JSON.stringify(SecurityGroupRules, null, 2));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [AuthorizeSecurityGroupIngress](#)를 참조하세요.

CreateKeyPair

다음 코드 예시는 CreateKeyPair의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { CreateKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates an ED25519 or 2048-bit RSA key pair with the specified name and in the
 * specified PEM or PPK format.
 * Amazon EC2 stores the public key and displays the private key for you to save to
 * a file.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new CreateKeyPairCommand({
    KeyName: keyName,
  });

  try {
    const { KeyMaterial, KeyName } = await client.send(command);
    console.log(KeyName);
    console.log(KeyMaterial);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidKeyPair.Duplicate") {
      console.warn(`${caught.message}. Try another key name.`);
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateKeyPair](#)를 참조하세요.

CreateLaunchTemplate

다음 코드 예시는 CreateLaunchTemplate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateLaunchTemplate](#)을 참조하세요.

CreateSecurityGroup

다음 코드 예시는 CreateSecurityGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { CreateSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates a security group.
 * @param {{ groupName: string, description: string }} options
 */
export const main = async ({ groupName, description }) => {
  const client = new EC2Client({});
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: groupName,
    // Up to 255 characters in length.
    Description: description,
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateSecurityGroup](#)을 참조하세요.

DeleteKeyPair

다음 코드 예시는 DeleteKeyPair의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DeleteKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes the specified key pair, by removing the public key from Amazon EC2.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new DeleteKeyPairCommand({
    KeyName: keyName,
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide the required value?`);
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteKeyPair](#)를 참조하세요.

DeleteLaunchTemplate

다음 코드 예시는 DeleteLaunchTemplate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
await client.send(
  new DeleteLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
  }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteLaunchTemplate](#)을 참조하세요.

DeleteSecurityGroup

다음 코드 예시는 DeleteSecurityGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DeleteSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes a security group.
 * @param {{ groupId: string }} options
 */
export const main = async ({ groupId }) => {
  const client = new EC2Client({});
  const command = new DeleteSecurityGroupCommand({
    GroupId: groupId,
```

```
});

try {
  await client.send(command);
  console.log("Security group deleted successfully.");
} catch (caught) {
  if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
    console.warn(`${caught.message}. Please provide a valid GroupId.`);
  } else {
    throw caught;
  }
}
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteSecurityGroup](#)을 참조하세요.

DescribeAddresses

다음 코드 예시는 DescribeAddresses의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DescribeAddressesCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Describes the specified Elastic IP addresses or all of your Elastic IP addresses.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
  const client = new EC2Client({});
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: [allocationId],
  });
```

```

try {
  const { Addresses } = await client.send(command);
  const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
  console.log("Elastic IP addresses:");
  console.log(addressList.join("\n"));
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "InvalidAllocationID.NotFound"
  ) {
    console.warn(`${caught.message}. Please provide a valid AllocationId.`);
  } else {
    throw caught;
  }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeAddresses](#) 참조하세요.

DescribeIamInstanceProfileAssociations

다음 코드 예시는 DescribeIamInstanceProfileAssociations의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeInstanceProfileAssociations](#)를 참조하세요.

DescribeImages

다음 코드 예시는 DescribeImages의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { EC2Client, paginateDescribeImages } from "@aws-sdk/client-ec2";

/**
 * Describes the specified images (AMIs, AKIs, and ARIs) available to you or all of
 * the images available to you.
 * @param {{ architecture: string, pageSize: number }} options
 */
export const main = async ({ architecture, pageSize }) => {
  pageSize = Number.parseInt(pageSize);
  const client = new EC2Client({});

  // 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 },
    {
      // 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: [architecture] }],
    },
  );
```

```

);

/**
 * @type {import('@aws-sdk/client-ec2').Image[]}
 */
const images = [];
let recordsScanned = 0;

try {
  for await (const page of paginator) {
    recordsScanned += pageSize;
    if (page.Images.length) {
      images.push(...page.Images);
      break;
    }
    console.log(
      `No matching image found yet. Searched ${recordsScanned} records.`,
    );
  }

  if (images.length) {
    console.log(
      `Found ${images.length} images:\n\n${images.map((image) =>
image.Name).join("\n")}\n`,
    );
  } else {
    console.log(
      `No matching images found. Searched ${recordsScanned} records.\n`,
    );
  }

  return images;
} catch (caught) {
  if (caught instanceof Error && caught.name === "InvalidParameterValue") {
    console.warn(`${caught.message}`);
    return [];
  }
  throw caught;
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeImages](#)를 참조하세요.

DescribeInstanceTypes

다음 코드 예시는 DescribeInstanceTypes의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { EC2Client, paginateDescribeInstanceTypes } from "@aws-sdk/client-ec2";

/**
 * Describes the specified instance types. By default, all instance types for the
 * current Region are described. Alternatively, you can filter the results.
 * @param {{ pageSize: string, supportedArch: string[], freeTier: boolean }} options
 */
export const main = async ({ pageSize, supportedArch, freeTier }) => {
  pageSize = Number.parseInt(pageSize);
  const client = new EC2Client({});

  // 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 },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          Values: supportedArch,
        },
        { Name: "free-tier-eligible", Values: [freeTier ? "true" : "false"] },
      ],
    },
  );

  try {
    /**
     * @type {import('@aws-sdk/client-ec2').InstanceTypeInfo[]}
```

```

    */
    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(
        `Memory size in MiB for matching instance types:\n\n${instanceTypes.map((it)
=> `${it.InstanceType}: ${it.MemoryInfo.SizeInMiB} MiB`).join("\n")}`,
    );
} catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        console.warn(`${caught.message}`);
        return [];
    }
    throw caught;
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeInstanceTypes](#)를 참조하세요.

DescribeInstances

다음 코드 예시는 DescribeInstances의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { EC2Client, paginateDescribeInstances } from "@aws-sdk/client-ec2";
```

```

/**
 * List all of your EC2 instances running with the provided architecture that
 * were launched in the past month.
 * @param {{ pageSize: string, architectures: string[] }} options
 */
export const main = async ({ pageSize, architectures }) => {
  pageSize = Number.parseInt(pageSize);
  const client = new EC2Client({});
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;

  const paginator = paginateDescribeInstances(
    {
      client,
      pageSize,
    },
    {
      Filters: [
        { Name: "architecture", Values: architectures },
        { Name: "instance-state-name", Values: ["running"] },
        {
          Name: "launch-time",
          Values: [launchTimePattern],
        },
      ],
    },
  );

  try {
    /**
     * @type {import('@aws-sdk/client-ec2').Instance[]}
     */
    const instanceList = [];
    for await (const page of paginator) {
      const { Reservations } = page;
      for (const reservation of Reservations) {
        instanceList.push(...reservation.Instances);
      }
    }
    console.log(

```

```

    `Running instances launched this month:\n\n${instanceList.map((instance) =>
instance.InstanceId).join("\n")}``,
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeInstances](#) 참조하세요.

DescribeKeyPairs

다음 코드 예시는 DescribeKeyPairs의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { DescribeKeyPairsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * List all key pairs in the current AWS account.
 * @param {{ dryRun: boolean }}
 */
export const main = async ({ dryRun }) => {
  const client = new EC2Client({});
  const command = new DescribeKeyPairsCommand({ DryRun: dryRun });

  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 (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
      console.log(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeKeyPairs](#)를 참조하세요.

DescribeRegions

다음 코드 예시는 DescribeRegions의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { DescribeRegionsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * List all available AWS regions.
 * @param {{ regionNames: string[], includeOptInRegions: boolean }} options
 */
export const main = async ({ regionNames, includeOptInRegions }) => {
  const client = new EC2Client({});
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions is true, even the regions that require opt-in will be
    returned.
    AllRegions: includeOptInRegions,
    // You can omit the Filters property if you want to get all regions.
    Filters: regionNames?.length
    ? [

```

```

    {
      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: regionNames,
    },
  ]
  : undefined,
});

try {
  const { Regions } = await client.send(command);
  const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
  console.log("Found regions:");
  console.log(regionsList.join("\n"));
} catch (caught) {
  if (caught instanceof Error && caught.name === "DryRunOperation") {
    console.log(`${caught.message}`);
  } else {
    throw caught;
  }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeRegions](#)를 참조하세요.

DescribeSecurityGroups

다음 코드 예시는 DescribeSecurityGroups의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";
```



```

/**
 * Describes the specified security groups or all of your security groups.
 * @param {{ groupIds: string[] }} options
 */
export const main = async ({ groupIds = [] }) => {
  const client = new EC2Client({});
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: groupIds,
  });

  try {
    const { SecurityGroups } = await client.send(command);
    const sgList = SecurityGroups.map(
      (sg) => `• ${sg.GroupName} (${sg.GroupId}): ${sg.Description}`,
    ).join("\n");
    if (sgList.length) {
      console.log(`Security groups:\n${sgList}`);
    } else {
      console.log("No security groups found.");
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else if (
      caught instanceof Error &&
      caught.name === "InvalidGroup.NotFound"
    ) {
      console.warn(caught.message);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeSecurityGroups](#)를 참조하세요.

DescribeSubnets

다음 코드 예시는 DescribeSubnets의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeSubnets](#)를 참조하세요.

DescribeVpcs

다음 코드 예시는 DescribeVpcs의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeVpcs](#)를 참조하세요.

DisassociateAddress

다음 코드 예시는 DisassociateAddress의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DisassociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Disassociate an Elastic IP address from an instance.
 * @param {{ associationId: string }} options
 */
export const main = async ({ associationId }) => {
  const client = new EC2Client({});
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
    retired.
    AssociationId: associationId,
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAssociationID.NotFound"
    ) {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
}
```

```

    }
  };

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DisassociateAddress](#)를 참조하세요.

MonitorInstances

다음 코드 예시는 MonitorInstances의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { EC2Client, MonitorInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * 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.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new MonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  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 (caught) {

```

```

    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        console.warn(`${caught.message}`);
    } else {
        throw caught;
    }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [MonitorInstances](#)를 참조하세요.

RebootInstances

다음 코드 예시는 RebootInstances의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { EC2Client, RebootInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Requests a reboot of the specified instances. This operation is asynchronous;
 * it only queues a request to reboot the specified instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
    const client = new EC2Client({});
    const command = new RebootInstancesCommand({
        InstanceIds: instanceIds,
    });

    try {
        await client.send(command);
        console.log("Instance rebooted successfully.");
    } catch (caught) {
        if (
            caught instanceof Error &&

```

```

        caught.name === "InvalidInstanceID.NotFound"
    ) {
        console.warn(
            `${caught.message}. Please provide the InstanceId of a valid instance to
reboot.` ,
        );
    } else {
        throw caught;
    }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [RebootInstances](#)를 참조하세요.

ReleaseAddress

다음 코드 예시는 ReleaseAddress의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { ReleaseAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Release an Elastic IP address.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
    const client = new EC2Client({});
    const command = new ReleaseAddressCommand({
        // You can also use PublicIp, but that is for EC2 classic which is being
        retired.
        AllocationId: allocationId,
    });

    try {

```

```

    await client.send(command);
    console.log("Successfully released address.");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationID.`);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ReleaseAddress](#)를 참조하세요.

ReplaceIamInstanceProfileAssociation

다음 코드 예시는 ReplaceIamInstanceProfileAssociation의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ReplaceIamInstanceProfileAssociation](#)을 참조하세요.

RunInstances

다음 코드 예시는 RunInstances의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { EC2Client, RunInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Create new EC2 instances.
 * @param {{
 *   keyName: string,
 *   securityGroupIds: string[],
 *   imageId: string,
 *   instanceType: import('@aws-sdk/client-ec2')._InstanceType,
 *   minCount?: number,
 *   maxCount?: number }} options
 */
export const main = async ({
  keyName,
  securityGroupIds,
  imageId,
  instanceType,
  minCount = "1",
  maxCount = "1",
}) => {
  const client = new EC2Client({});
  minCount = Number.parseInt(minCount);
  maxCount = Number.parseInt(maxCount);
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: keyName,
    // Your security group.
    SecurityGroupIds: securityGroupIds,
    // An Amazon Machine Image (AMI). There are multiple ways to search for AMIs.
    // For more information, see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html
  });
  const data = await client.send(command);
  console.log(data);
}
```



```

    ImageId: imageId,
    // An instance type describing the resources provided to your instance. There
    are multiple
    // ways to search for instance types. For more information see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-discovery.html
    InstanceType: instanceType,
    // Availability Zones have capacity limitations that may impact your ability to
    launch instances.
    // The `RunInstances` operation will only succeed if it can allocate at least
    the `MinCount` of instances.
    // However, EC2 will attempt to launch up to the `MaxCount` of instances, even
    if the full request cannot be satisfied.
    // If you need a specific number of instances, use `MinCount` and `MaxCount` set
    to the same value.
    // If you want to launch up to a certain number of instances, use `MaxCount` and
    let EC2 provision as many as possible.
    // If you require a minimum number of instances, but do not want to exceed a
    maximum, use both `MinCount` and `MaxCount`.
    MinCount: minCount,
    MaxCount: maxCount,
  });

  try {
    const { Instances } = await client.send(command);
    const instanceList = Instances.map(
      (instance) => `• ${instance.InstanceId}`,
    ).join("\n");
    console.log(`Launched instances:\n${instanceList}`);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceCountExceeded") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [RunInstances](#)를 참조하세요.

StartInstances

다음 코드 예시는 StartInstances의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { EC2Client, StartInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Starts an Amazon EBS-backed instance that you've previously stopped.
 * @param {{ instanceIds }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StartInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [StartInstances](#)를 참조하세요.

StopInstances

다음 코드 예시는 StopInstances의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { EC2Client, StopInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Stop one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StopInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(` ${caught.message} `);
    } else {

```

```
        throw caught;
    }
}
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [StopInstances](#)를 참조하세요.

TerminateInstances

다음 코드 예시는 TerminateInstances의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { EC2Client, TerminateInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Terminate one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
    const client = new EC2Client({});
    const command = new TerminateInstancesCommand({
        InstanceIds: instanceIds,
    });

    try {
        const { TerminatingInstances } = await client.send(command);
        const instanceList = TerminatingInstances.map(
            (instance) => ` • ${instance.InstanceId}`,
        );
        console.log("Terminating instances:");
        console.log(instanceList.join("\n"));
    } catch (caught) {
```

```

    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [TerminateInstances](#)를 참조하세요.

UnmonitorInstances

다음 코드 예시는 UnmonitorInstances의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { EC2Client, UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Turn off detailed monitoring for the selected instance.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new UnmonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
  }
};

```

```

const instanceMonitoringsList = InstanceMonitorings.map(
  (im) =>
    ` • Detailed monitoring state for ${im.InstanceId} is
    ${im.Monitoring.State}.`,
  );
console.log("Monitoring status:");
console.log(instanceMonitoringsList.join("\n"));
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "InvalidInstanceID.NotFound"
  ) {
    console.warn(`${caught.message}`);
  } else {
    throw caught;
  }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UnmonitorInstances](#)를 참조하세요.

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.
- Auto Scaling 그룹의 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 파라미터를 업데이트하여 요청 및 상태 확인에 대한 웹 서버 응답을 제어합니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}
```

모든 리소스를 배포하기 위한 단계를 생성합니다.

```
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 { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 () => {
    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,
        },
      }),
    );
  })),
  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) => {
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  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) => {
  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"] },
    ],
  )),
);
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) => {
  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,
    }),
  );
  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) => {
    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] },
    );
  }),
  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) => {
    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 },

```

```

    ],
  })),
);
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) => {
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
}),
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),
      );
    }
    return MESSAGES.noIpRules;
  },
),
new ScenarioInput(
  "shouldAddInboundRule",
  /**
   * @param {{ myIpRules: any[] }} state

```

```

    */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      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);
    }
    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,
    );
  }
}),
saveState,
];

```

데모를 실행하기 위한 단계를 생성합니다.

```

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) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  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: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      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];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [

```

```

        { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
      ],
    })),
  );
state.instanceProfileAssociationId =
  IamInstanceProfileAssociations[0].AssociationId;
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    })),
  );

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",
  }),
);
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
  }),
);
```



```
    return InstanceProfile;
  }
}
```

모든 리소스를 폐기하는 단계를 생성합니다.

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} 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 { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 = [
  loadState,
  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(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${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(
          "${KEY_PAIR_NAME}",
          NAMES.keyPairName,
        );
      }
      return MESSAGES.deletedKeyPair.replace(
        "${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("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
      }
      return MESSAGES.detachedPolicyFromRole
        .replace("${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,
            );
        }
        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);
  }
  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(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
  return MESSAGES.deletedInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  );
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }

```

```

    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  })),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
    return MESSAGES.deletedInstanceProfile.replace(
      "${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(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  })),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(

```

```

        new DeleteLaunchTemplateCommand({
            LaunchTemplateName: NAMES.launchTemplateName,
        }),
    );
} catch (e) {
    state.deleteLaunchTemplateError = e;
}
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
        console.error(state.deleteLaunchTemplateError);
        return MESSAGES.deleteLaunchTemplateError.replace(
            "${LAUNCH_TEMPLATE_NAME}",
            NAMES.launchTemplateName,
        );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
    );
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
        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.");
            }
        });
    } catch (e) {
        state.deleteLoadBalancerError = e;
    }
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
        console.error(state.deleteLoadBalancerError);
        return MESSAGES.deleteLoadBalancerError.replace(

```

```

        "${LB_NAME}",
        NAMES.loadBalancerName,
    );
}
return MESSAGES.deletedLoadBalancer.replace(
    "${LB_NAME}",
    NAMES.loadBalancerName,
);
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    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;
    }
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
        console.error(state.deleteLoadBalancerTargetGroupError);
        return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
            "${TARGET_GROUP_NAME}",
            NAMES.loadBalancerTargetGroupName,
        );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${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);
  }
  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);
  }
  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");
        }
        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,
    );
}
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,
        );
    }
    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(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  })),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
      state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    }
  })
);

```

```

    );
  }
  return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
}),
];

/**
 * @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;
    }
    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](#)

SDK for JavaScript(v3)를 사용한 Elastic Load Balancing 버전 2 예제

다음 코드 예제에서는 Elastic Load Balancing - 버전 2와 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)
- [시나리오](#)

시작하기

Hello Elastic Load Balancing

다음 코드 예제에서는 Elastic Load Balancing 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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 "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeLoadBalancers](#)를 참조하세요.

작업

CreateListener

다음 코드 예시는 CreateListener의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 },
    ],
  }),
);

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateListener](#)를 참조하세요.

CreateLoadBalancer

다음 코드 예시는 CreateLoadBalancer의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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] },
);
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateLoadBalancer](#)를 참조하세요.

CreateTargetGroup

다음 코드 예시는 CreateTargetGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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,
  }),
);
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateTargetGroup](#)를 참조하세요.

DeleteLoadBalancer

다음 코드 예시는 DeleteLoadBalancer의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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.");
  }
});
```

```
}  
});
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteLoadBalancer](#)를 참조하세요.

DeleteTargetGroup

다음 코드 예시는 DeleteTargetGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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;  
}
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteTargetGroup](#)을 참조하세요.

DescribeLoadBalancers

다음 코드 예시는 DescribeLoadBalancers의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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 "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeLoadBalancers](#)를 참조하세요.

DescribeTargetGroups

다음 코드 예시는 DescribeTargetGroups의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeTargetGroups](#)를 참조하세요.

DescribeTargetHealth

다음 코드 예시는 DescribeTargetHealth의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeTargetHealth](#)를 참조하세요.

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.
- Auto Scaling 그룹의 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 파라미터를 업데이트하여 요청 및 상태 확인에 대한 웹 서버 응답을 제어합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}

```



```
});
}
```

모든 리소스를 배포하기 위한 단계를 생성합니다.

```
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 { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utlis/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 () => {
      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,
          },
        }),
      );
    }),
    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) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(

```

```

        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
    ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
        new CreateTargetGroupCommand({
            Name: NAMES.loadBalancerTargetGroupName,
            Protocol: "HTTP",
            Port: 80,
            HealthCheckPath: "/healthcheck",
            HealthCheckIntervalSeconds: 10,
            HealthCheckTimeoutSeconds: 5,
            HealthyThresholdCount: 2,
            UnhealthyThresholdCount: 2,
            VpcId: state.defaultVpc,
        })),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
    ),
),
new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
    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] },
    );
  }},
  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) => {
    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 },
        ],
      })
    );
    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) => {
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
}),
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),
      );
    }
    return MESSAGES.noIpRules;
  },
),
new ScenarioInput(
  "shouldAddInboundRule",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return false;
    }
    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);
  }
  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,
    );
  }
}),
saveState,

```

```
];
```

데모를 실행하기 위한 단계를 생성합니다.

```
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) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(

```



```

    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  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: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {

```

```

    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      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];
        const ec2Client = new EC2Client({});
        const { IamInstanceProfileAssociations } = await ec2Client.send(
            new DescribeIamInstanceProfileAssociationsCommand({
                Filters: [
                    { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
                ],
            }),
        );
        state.instanceProfileAssociationId =
            IamInstanceProfileAssociations[0].AssociationId;
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            ec2Client.send(
                new ReplaceIamInstanceProfileAssociationCommand({
                    AssociationId: state.instanceProfileAssociationId,
                    IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
                }),
            ),
        );

        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",
    })),
);
const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    })),
);
await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
    new AddRoleToInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
    })),
);

return InstanceProfile;
}

```

모든 리소스를 폐기하는 단계를 생성합니다.

```

import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
    EC2Client,
    DeleteKeyPairCommand,
    DeleteLaunchTemplateCommand,
    RevokeSecurityGroupIngressCommand,
} 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 { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 = [
  loadState,
  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(
      "${TABLE_NAME}",
      NAMES.tableName,
    );
  }
  return MESSAGES.deletedTable.replace("${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,
    );
  }
  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("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
  return MESSAGES.detachedPolicyFromRole
    .replace("${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(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }
  return MESSAGES.deletedPolicy.replace(
    "${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);
  }
  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(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
  return MESSAGES.deletedInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  );
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
  return MESSAGES.deletedInstanceProfile.replace(
    "${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(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  })),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  })),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",

```

```

    NAMES.launchTemplateName,
  );
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
    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.");
      }
    });
  } catch (e) {
    state.deleteLoadBalancerError = e;
  }
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
  return MESSAGES.deletedLoadBalancer.replace(
    "${LB_NAME}",
    NAMES.loadBalancerName,
  );
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  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;
  }
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }
  return MESSAGES.deletedLoadBalancerTargetGroup.replace(
    "${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);
  }
}

```

```

    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);
    }
    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}", "AmazonSSManagedInstanceCore");
    }
    return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSManagedInstanceCore");
    })),
    new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
        try {
            const iamClient = new IAMClient({});
            await iamClient.send(
                new DeleteInstanceProfileCommand({
                    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
                }),
            );
        } catch (e) {
            state.deleteSsmOnlyInstanceProfileError = e;
        }
    })),
    new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
        if (state.deleteSsmOnlyInstanceProfileError) {
            console.error(state.deleteSsmOnlyInstanceProfileError);
            return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
                "${INSTANCE_PROFILE_NAME}",
                NAMES.ssmOnlyInstanceProfileName,
            );
        }
        return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
            "${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(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
    return MESSAGES.deletedSsmOnlyPolicy.replace(
      "${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(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  })),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (

```

```

    /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
    state,
  ) => {
    const ec2Client = new EC2Client({});

    try {
      await ec2Client.send(
        new RevokeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    } catch (e) {
      state.revokeSecurityGroupIngressError = e;
    }
  },
),
new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
  if (state.revokeSecurityGroupIngressError) {
    console.error(state.revokeSecurityGroupIngressError);
    return MESSAGES.revokeSecurityGroupIngressError.replace(
      "${IP}",
      state.myIp,
    );
  }
  return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
}),
];

/**
 * @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;
    }
    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](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

AWS Entity Resolution SDK for JavaScript(v3)를 사용한 예제

다음 코드 예제에서는 AWS SDK for JavaScript (v3)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS Entity Resolution.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)

시작하기

안녕하세요 AWS Entity Resolution

다음 코드 예제에서는 AWS Entity Resolution를 사용하여 시작하는 방법을 보여 줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.


```
import { fileURLToPath } from "node:url";
import {
  EntityResolutionClient,
  ListMatchingWorkflowsCommand,
} from "@aws-sdk/client-entityresolution";

export const main = async () => {
  const region = "eu-west-1";
  const erClient = new EntityResolutionClient({ region: region });
  try {
    const command = new ListMatchingWorkflowsCommand({});
    const response = await erClient.send(command);
    const workflowSummaries = response.workflowSummaries;
    for (const workflowSummary of workflowSummaries) {
      console.log(`Attribute name: ${workflowSummaries[0].workflowName} `);
    }
    if (workflowSummaries.length === 0) {
      console.log("No matching workflows found.");
    }
  } catch (error) {
    console.error(
      `An error occurred in listing the workflow summaries: ${error.message} \n
      Exiting program.`
    );
    return;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListMatchingWorkflows](#)를 참조하세요.

작업

CreateMatchingWorkflow

다음 코드 예시는 CreateMatchingWorkflow의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  CreateMatchingWorkflowCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const createMatchingWorkflowParams = {
    roleArn: `${data.inputs.roleArn}`,
    workflowName: `${data.inputs.workflowName}`,
    description: "Created by using the AWS SDK for JavaScript (v3).",
    inputSourceConfig: [
      {
        inputSourceARN: `${data.inputs.JSONinputSourceARN}`,
        schemaName: `${data.inputs.schemaNameJson}`,
        applyNormalization: false,
      },
      {
        inputSourceARN: `${data.inputs.CSVinputSourceARN}`,
        schemaName: `${data.inputs.schemaNameCSV}`,
        applyNormalization: false,
      },
    ],
    outputSourceConfig: [
      {
        outputS3Path: `s3://${data.inputs.myBucketName}/eroutput`,
      },
    ],
  };
};
```

```

        output: [
            {
                name: "id",
            },
            {
                name: "name",
            },
            {
                name: "email",
            },
            {
                name: "phone",
            },
        ],
        applyNormalization: false,
    },
],
resolutionTechniques: { resolutionType: "ML_MATCHING" },
};
try {
    const command = new CreateMatchingWorkflowCommand(
        createMatchingWorkflowParams,
    );
    const response = await erClient.send(command);

    console.log(
        `Workflow created successfully.\n The workflow ARN is:
        ${response.workflowArn}`,
    );
} catch (caught) {
    console.error(caught.message);
    throw caught;
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateMatchingWorkflow](#)를 참조하세요.

CreateSchemaMapping

다음 코드 예시는 CreateSchemaMapping의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  CreateSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const createSchemaMappingParamsJson = {
    schemaName: `${data.inputs.schemaNameJson}`,
    mappedInputFields: [
      {
        fieldName: "id",
        type: "UNIQUE_ID",
      },
      {
        fieldName: "name",
        type: "NAME",
      },
      {
        fieldName: "email",
        type: "EMAIL_ADDRESS",
      },
    ],
  };
  const createSchemaMappingParamsCSV = {
    schemaName: `${data.inputs.schemaNameCSV}`,
```

```
mappedInputFields: [
  {
    fieldName: "id",
    type: "UNIQUE_ID",
  },
  {
    fieldName: "name",
    type: "NAME",
  },
  {
    fieldName: "email",
    type: "EMAIL_ADDRESS",
  },
  {
    fieldName: "phone",
    type: "PROVIDER_ID",
    subType: "STRING",
  },
],
};
try {
  const command = new CreateSchemaMappingCommand(
    createSchemaMappingParamsJson,
  );
  const response = await erClient.send(command);
  console.log("The JSON schema mapping name is ", response.schemaName);
} catch (error) {
  console.log("error ", error.message);
}
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateSchemaMapping](#)을 참조하세요.

DeleteMatchingWorkflow

다음 코드 예시는 DeleteMatchingWorkflow의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  DeleteMatchingWorkflowCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  try {
    const deleteWorkflowParams = {
      workflowName: `${data.inputs.workflowName}`,
    };
    const command = new DeleteMatchingWorkflowCommand(deleteWorkflowParams);
    const response = await erClient.send(command);
    console.log("Workflow deleted successfully!", response);
  } catch (error) {
    console.log("error ", error);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteMatchingWorkflow](#)를 참조하세요.

DeleteSchemaMapping

다음 코드 예시는 DeleteSchemaMapping의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  DeleteSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const deleteSchemaMapping = {
    schemaName: `${data.inputs.schemaNameJson}`,
  };
  try {
    const command = new DeleteSchemaMappingCommand(deleteSchemaMapping);
    const response = await erClient.send(command);
    console.log("Schema mapping deleted successfully. ", response);
  } catch (error) {
    console.log("error ", error);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteSchemaMapping](#)를 참조하세요.

GetMatchingJob

다음 코드 예시는 GetMatchingJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  GetMatchingJobCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  async function getInfo() {
    const getJobInfoParams = {
      workflowName: `${data.inputs.workflowName}`,
      jobId: `${data.inputs.jobId}`,
    };
    try {
      const command = new GetMatchingJobCommand(getJobInfoParams);
      const response = await erClient.send(command);
      console.log(`Job status: ${response.status}`);
    } catch (error) {
      console.log("error ", error.message);
    }
  }
};
```


- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetMatchingJob](#)을 참조하세요.

GetSchemaMapping

다음 코드 예시는 GetSchemaMapping의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  GetSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const getSchemaMappingJsonParams = {
    schemaName: `${data.inputs.schemaNameJson}`,
  };
  try {
    const command = new GetSchemaMappingCommand(getSchemaMappingJsonParams);
    const response = await erClient.send(command);
    console.log(response);
    console.log(
      `Schema mapping for the JSON data:\n ${response.mappedInputFields[0]}`,
    );
  }
}
```

```

    console.log("Schema mapping ARN is: ", response.schemaArn);
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetSchemaMapping](#)을 참조하세요.

ListSchemaMappings

다음 코드 예시는 ListSchemaMappings의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  ListSchemaMappingsCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  async function getInfo() {
    const listSchemaMappingsParams = {
      workflowName: `${data.inputs.workflowName}`,

```

```

    jobId: `${data.inputs.jobId}`,
  };
  try {
    const command = new ListSchemaMappingsCommand(listSchemaMappingsParams);
    const response = await erClient.send(command);
    const noOfSchemas = response.schemaList.length;
    for (let i = 0; i < noOfSchemas; i++) {
      console.log(
        `Schema Mapping Name: ${response.schemaList[i].schemaName} `,
      );
    }
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListSchemaMappings](#)을 참조하세요.

StartMatchingJob

다음 코드 예시는 StartMatchingJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//The default inputs for this demo are read from the ../inputs.json.
```

```

import { fileURLToPath } from "node:url";
import {
  StartMatchingJobCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

```

```
const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const matchingJobOfWorkflowParams = {
    workflowName: `${data.inputs.workflowName}`,
  };
  try {
    const command = new StartMatchingJobCommand(matchingJobOfWorkflowParams);
    const response = await erClient.send(command);
    console.log(`Job ID: ${response.jobID} \n
The matching job was successfully started.`);
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [StartMatchingJob](#)을 참조하세요.

TagResource

다음 코드 예시는 TagResource의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  TagResourceCommand,
```

```

    EntityResolutionClient,
  } from "@aws-sdk/client-entityresolution";
  import data from "../inputs.json" with { type: "json" };

  const region = "eu-west-1";
  const erClient = new EntityResolutionClient({ region: region });

  export const main = async () => {
    const tagResourceCommandParams = {
      resourceArn: `${data.inputs.schemaArn}`,
      tags: {
        tag1: "tag1Value",
        tag2: "tag2Value",
      },
    };
    try {
      const command = new TagResourceCommand(tagResourceCommandParams);
      const response = await erClient.send(command);
      console.log("Successfully tagged the resource.");
    } catch (caught) {
      console.error(caught.message);
      throw caught;
    }
  };

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [TagResource](#)를 참조하세요.

SDK for JavaScript (v3)를 사용한 EventBridge 예제

다음 코드 예제에서는 EventBridge와 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)
- [시나리오](#)

작업

PutEvents

다음 코드 예시는 PutEvents의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [PutEvents](#)를 참조하세요.

PutRule

다음 코드 예시는 PutRule의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [PutRule](#)을 참조하세요.

PutTargets

다음 코드 예시는 PutTargets의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [PutTargets](#)를 참조하세요.

시나리오

예약된 이벤트를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon EventBridge 예약 이벤트에서 호출된 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS Lambda 함수를 호출하는 Amazon EventBridge 예약 이벤트를 생성하는 방법을 보여줍니다. Lambda 함수가 간접 호출될 때 cron 표현식을 사용하여 일정을 예약하도록 EventBridge를 구성합니다. 이 예제에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

SDK for JavaScript(v3)를 사용한 Amazon Glacier 예제

다음 코드 예제에서는 Amazon Glacier에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

CreateVault

다음 코드 예시는 CreateVault의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 생성합니다.

```
// 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 안내서](#)를 참조하세요.
- API 세부 정보는AWS SDK for JavaScript API 참조의 [CreateVault](#)를 참조하세요.

UploadArchive

다음 코드 예시는 UploadArchive의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 업로드합니다.

```
// 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UploadArchive](#)를 참조하세요.

AWS Glue SDK for JavaScript(v3)를 사용한 예제

다음 코드 예제에서는 AWS SDK for JavaScript (v3)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS Glue.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)

시작하기

안녕하세요 AWS Glue

다음 코드 예제에서는 AWS Glue를 사용하여 시작하는 방법을 보여 줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListJobs](#)를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 퍼블릭 Amazon S3 버킷을 크롤링하고 CSV 형식의 메타데이터 데이터베이스를 생성하는 크롤러를 생성합니다.
- 의 데이터베이스 및 테이블에 대한 정보를 나열합니다 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 참조하세요](#).

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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;
    }
};

/**
 * @param {{ createCrawler: import('../../actions/create-crawler.js').createCrawler }} actions
 */
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 };
  };

```

의 데이터베이스 및 테이블에 대한 정보를 나열합니다 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 };
  };

/**
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-glue').GetTablesCommandOutput> }} config
 */

```

```
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":
    case "ERROR":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "SUCCEEDED":
      return;
    default:
      break;
  }
}

```

```

    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);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput>}
getJobRun
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunsCommandOutput>}
getJobRuns
 */

/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName

```

```

* @param {string} jobRunId
*/
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
 *
 * @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
 */
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (/** @type { Context } */ 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);
};

/**
 *
 * @param {import('../../actions/delete-job.js').deleteJobFn} deleteJobFn
 * @param {string[]} jobNames
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
   * @type {{ selectedJobNames: string[] }}
   */
  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" });
  }
};

/**
 * @param {{
 *   listJobs: import('.././../actions/list-jobs.js').listJobs,
 *   deleteJob: import('.././../actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

/**
 * @param {import('.././../actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

```

```

/**
 * @param {{
 *   getTables: import('.././../actions/get-tables.js').getTables,
 *   deleteTable: import('.././../actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any> }}} context
   */
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null }),
    );

    if (TableList && TableList.length > 0) {
      /**
       * @type {{ tableNames: string[] }}
       */
      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 };
  };

/**
 * @param {import('.././../actions/delete-database.js').deleteDatabase}
 * deleteDatabase
 * @param {string[]} databaseNames
 */

```

```

const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('.././../actions/get-databases.js').getDatabases
 *   deleteDatabase: import('.././../actions/delete-database.js').deleteDatabase
 * }} config
 */
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any> } } } context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbName: string[] }} */
      const { dbName } = 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 (dbName.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbName);
        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](#)

작업

CreateCrawler

다음 코드 예시는 CreateCrawler의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateCrawler](#)를 참조하세요.

CreateJob

다음 코드 예시는 CreateJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateJob](#)을 참조하세요.

DeleteCrawler

다음 코드 예시는 DeleteCrawler의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteCrawler](#)를 참조하세요.

DeleteDatabase

다음 코드 예시는 DeleteDatabase의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteDatabase](#)를 참조하세요.

DeleteJob

다음 코드 예시는 DeleteJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteJob](#)을 참조하세요.

DeleteTable

다음 코드 예시는 DeleteTable의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteTable](#)을 참조하세요.

GetCrawler

다음 코드 예시는 GetCrawler의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetCrawler](#)를 참조하세요.

GetDatabase

다음 코드 예시는 GetDatabase의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetDatabase](#)를 참조하세요.

GetDatabases

다음 코드 예시는 GetDatabases의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetDatabases](#)를 참조하세요.

GetJob

다음 코드 예시는 GetJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetJob](#)을 참조하세요.

GetJobRun

다음 코드 예시는 GetJobRun의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetJobRun](#)을 참조하세요.

GetJobRuns

다음 코드 예시는 GetJobRuns의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetJobRuns](#)를 참조하세요.

GetTables

다음 코드 예시는 GetTables의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetTables](#)를 참조하세요.

ListJobs

다음 코드 예시는 ListJobs의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListJobs](#)를 참조하세요.

StartCrawler

다음 코드 예시는 StartCrawler의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [StartCrawler](#)를 참조하세요.

StartJobRun

다음 코드 예시는 StartJobRun의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [StartJobRun](#)을 참조하세요.

SDK for JavaScript (v3)를 사용한 HealthImaging 예

다음 코드 예제에서는 HealthImaging과 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)
- [시나리오](#)

시작하기

HealthImaging 시작

다음 코드 예제에서는 HealthImaging을 사용하여 시작하는 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [ListDatastores](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

작업

CopyImageSet

다음 코드 예시는 CopyImageSet의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

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.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = "",
  force = false,
  copySubsets = [],
) => {
  try {
    const params = {
      datastoreId: datastoreId,
      sourceImageSetId: imageSetId,
      copyImageSetInformation: {
        sourceImageSet: { latestVersionId: sourceVersionId },
      },
      force: force,
    };
    if (destinationImageSetId !== "" && destinationVersionId !== "") {
      params.copyImageSetInformation.destinationImageSet = {
        imageSetId: destinationImageSetId,
        latestVersionId: destinationVersionId,
      };
    }

    if (copySubsets.length > 0) {
```

```

let copySubsetsJson;
copySubsetsJson = {
  SchemaVersion: 1.1,
  Study: {
    Series: {
      imageSetId: {
        Instances: {},
      },
    },
  },
};

for (let i = 0; i < copySubsets.length; i++) {
  copySubsetsJson.Study.Series.imageSetId.Instances[copySubsets[i]] = {};
}

params.copyImageSetInformation.dicomCopies = copySubsetsJson;
}

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: 'xxxxxxxxxxxxxx',
//   destinationImageSetProperties: {
//     createdAt: 2023-09-27T19:46:21.824Z,
//     imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxxxxxx',
//     imageSetState: 'LOCKED',
//     imageSetWorkflowStatus: 'COPYING',
//     latestVersionId: '1',
//     updatedAt: 2023-09-27T19:46:21.824Z
//   },
//   sourceImageSetProperties: {

```

```
//          createdAt: 2023-09-22T14:49:26.427Z,
//          imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxx',
//          imageSetId: 'xxxxxxxxxxxxx',
//          imageSetState: 'LOCKED',
//          imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
//          latestVersionId: '4',
//          updatedAt: 2023-09-27T19:46:21.824Z
//      }
// }
return response;
} catch (err) {
    console.error(err);
}
};
```

대상 없이 이미지 세트를 복사합니다.

```
await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
);
```

대상이 있는 이미지 세트를 복사합니다.

```
await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
    "12345678901234567890123456789012",
    "1",
    false,
);
```

이미지 세트의 하위 집합을 대상 위치에 강제로 복사합니다.

```
await copyImageSet(
  "12345678901234567890123456789012",
  "12345678901234567890123456789012",
  "1",
  "12345678901234567890123456789012",
  "1",
  true,
  ["12345678901234567890123456789012", "11223344556677889900112233445566"],
);
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CopyImageSet](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CreateDatastore

다음 코드 예시는 CreateDatastore의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
```

```
//      extendedRequestId: undefined,
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//    datastoreStatus: 'CREATING'
//  }
  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateDatastore](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DeleteDatastore

다음 코드 예시는 DeleteDatastore의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //     extendedRequestId: undefined,
```

```
//          cfId: undefined,
//          attempts: 1,
//          totalRetryDelay: 0
//      },
//      datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      datastoreStatus: 'DELETING'
//  }

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteDatastore](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DeleteImageSet

다음 코드 예시는 DeleteImageSet의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
```

```

);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   datastoreId: 'xxxxxxxxxxxxxxxx',
//   imageSetId: 'xxxxxxxxxxxxxxxx',
//   imageSetState: 'LOCKED',
//   imageSetWorkflowStatus: 'DELETING'
// }
return response;
};

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteImageSet](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

GetDICOMImportJob

다음 코드 예시는 GetDICOMImportJob의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```

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 = "xxxxxxxxxxxxxxxxxxxxxx",
    jobId = "xxxxxxxxxxxxxxxxxxxxxx",
  ) => {
    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: 'xxxxxxxxxxxxxxxxxxxxxx',
    //     endedAt: 2023-09-19T17:29:21.753Z,
    //     inputS3Uri: 's3://healthimaging-source/CTStudy/',
    //     jobId: 'xxxxxxxxxxxxxxxxxxxxxx',
    //     jobName: 'job_1',
    //     jobStatus: 'COMPLETED',
    //     outputS3Uri: 's3://health-imaging-dest/
    output_ct/'xxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxx'/',
    //     submittedAt: 2023-09-19T17:27:25.143Z
    //   }
    // }

    return response;
  };

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [GetDICOMImportJob](#)을 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

GetImageFrame

다음 코드 예시는 GetImageFrame의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 * image frame.
 * @param {string} datastoreId - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID",
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    }),
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
  //   },
  // }
```

```
//      extendedRequestId: undefined,
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    contentType: 'application/octet-stream',
//    imageFrameBlob: <ref *1> IncomingMessage {}
//  }
return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [GetImageFrame](#)을 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

GetImageSet

다음 코드 예시는 GetImageSet의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
  imageSetVersion = "",
) => {
  const 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: 'xxxxxxxxxxxxxxxxxx',
//     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxxxxxx',
//     imageSetState: 'ACTIVE',
//     imageSetWorkflowStatus: 'CREATED',
//     updatedAt: 2023-09-22T14:49:26.427Z,
//     versionId: '1'
// }

return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [getImageSet](#)를 참조하세요.

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배우보세요.

GetImageSetMetadata

다음 코드 예시는 GetImageSetMetadata의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

이미지 세트 메타데이터를 가져오는 유틸리티 함수입니다.

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "node:fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxxxx",
  versionID = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params),
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   contentType: 'application/json',
  //   contentEncoding: 'gzip',
  // }
```

```
//      imageSetMetadataBlob: <ref *1> IncomingMessage {}  
// }  
  
return response;  
};
```

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);  
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetImageSetMetadata](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

ListDICOMImportJobs

다음 코드 예시는 ListDICOMImportJobs의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  const 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: 'xxxxxxxxxxxxxxxxxxxxxx',
  //       endedAt: 2023-09-22T14:49:51.351Z,
```

```
//          jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//          jobName: 'test-1',
//          jobStatus: 'COMPLETED',
//          submittedAt: 2023-09-22T14:48:45.767Z
// }
// ]}

return jobSummaries;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [ListDICOMImportJobs](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

ListDatastores

다음 코드 예시는 ListDatastores의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
```



```

    for await (const page of paginator) {
      // Each page contains a list of `jobSummaries`. The list is truncated if is
      // larger than `pageSize`.
      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/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //       datastoreName: 'my_datastore',
    //       datastoreStatus: 'ACTIVE',
    //       updatedAt: 2023-08-04T18:49:54.429Z
    //     }
    //     ...
    //   ]
    // }

    return datastoreSummaries;
  };

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [ListDatastores](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

ListImageSetVersions

다음 코드 예시는 ListImageSetVersions의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams,
  );

  const 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: 'xxxxxxxxxxxxxxxxxxxxxxxx',
//              imageSetState: 'ACTIVE',
//              versionId: '1'
//          }]
// }
return imageSetPropertiesList;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [ListImageSetVersions](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

ListTagsForResource

다음 코드 예시는 ListTagsForResource의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi",
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
};
```

```
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   tags: { Deployment: 'Development' }
// }

return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [ListTagsForResource](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

SearchImageSets

다음 코드 예시는 SearchImageSets의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

이미지 세트 검색을 위한 유틸리티 함수.

```
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.
 * @param { import('@aws-sdk/client-medical-imaging').Sort } sort - The search
  criteria sort.
 */
```

```

export const searchImageSets = async (
  datastoreId = "xxxxxxxx",
  searchCriteria = {},
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {
    datastoreId: datastoreId,
    searchCriteria: searchCriteria,
  };

  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 searchCriteria = {
    filters: [
      {
        values: [{ DICOMPatientId: "1234567" }],
        operator: "EQUAL",
      },
    ],
  };

  await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
  console.error(err);
}
```

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [
      {
        values: [
          {
            DICOMStudyDateAndTime: {
              DICOMStudyDate: "19900101",
              DICOMStudyTime: "000000",
            },
          },
          {
            DICOMStudyDateAndTime: {
              DICOMStudyDate: "20230901",
              DICOMStudyTime: "000000",
            },
          },
        ],
      },
    ],
  };

  await searchImageSets(datastoreId, searchCriteria);
}
```

```

        ],
        operator: "BETWEEN",
      },
    ],
  };

  await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
  console.error(err);
}

```

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```

const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [
      {
        values: [
          { createdAt: new Date("1985-04-12T23:20:50.52Z") },
          { createdAt: new Date() },
        ],
        operator: "BETWEEN",
      },
    ],
  };

  await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
  console.error(err);
}

```

사용 사례 4: DICOMSeriesInstanceUID 필드에는 EQUAL 연산자를 적용하고 updatedAt 필드에는 BETWEEN 연산자를 적용하며, 응답은 updatedAt 필드를 기준으로 오름차순(ASC) 정렬합니다.

```

const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [

```

```

    {
      values: [
        { updatedAt: new Date("1985-04-12T23:20:50.52Z") },
        { updatedAt: new Date() },
      ],
      operator: "BETWEEN",
    },
    {
      values: [
        {
          DICOMSeriesInstanceUID:
            "1.1.123.123456.1.12.1.1234567890.1234.12345678.123",
        },
      ],
      operator: "EQUAL",
    },
  ],
  sort: {
    sortOrder: "ASC",
    sortField: "updatedAt",
  },
};

await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
  console.error(err);
}

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [SearchImageSets](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

StartDICOMImportJob

다음 코드 예시는 StartDICOMImportJob의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```

import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/",
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   jobStatus: 'SUBMITTED',
  //   submittedAt: 2023-09-22T14:48:45.767Z
  // }

```

```
// }
return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [StartDICOMImportJob](#)을 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

TagResource

다음 코드 예시는 TagResource의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 * - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {},
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
```

```
//      extendedRequestId: undefined,
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    }
//  }

return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [TagResource](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

UntagResource

다음 코드 예시는 UntagResource의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
  xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
```

```
//      '$metadata': {
//          httpStatusCode: 204,
//          requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
//          extendedRequestId: undefined,
//          cfId: undefined,
//          attempts: 1,
//          totalRetryDelay: 0
//      }
//  }

return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [UntagResource](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

UpdateImageSetMetadata

다음 코드 예시는 UpdateImageSetMetadata의 사용 방법을 보여줍니다.

SDK for JavaScript(v3)

```
import { UpdateImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
```

```

    force = false,
  ) => {
    try {
      const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
          datastoreId: datastoreId,
          imageSetId: imageSetId,
          latestVersionId: latestVersionId,
          updateImageSetMetadataUpdates: updateMetadata,
          force: force,
        }),
      );
      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;
    } catch (err) {
      console.error(err);
    }
  };

```

사용 사례 1: 속성을 삽입 또는 업데이트하고 강제로 업데이트합니다.

```

const insertAttributes = JSON.stringify({
  SchemaVersion: 1.1,
  Study: {
    DICOM: {

```

```

        StudyDescription: "CT CHEST",
    },
},
});

const updateMetadata = {
  DICOMUpdates: {
    updatableAttributes: new TextEncoder().encode(insertAttributes),
  },
};

await updateImageSetMetadata(
  datastoreID,
  imageSetID,
  versionID,
  updateMetadata,
  true,
);

```

사용 사례 2: 속성을 제거합니다.

```

// Attribute key and value must match the existing attribute.
const remove_attribute = JSON.stringify({
  SchemaVersion: 1.1,
  Study: {
    DICOM: {
      StudyDescription: "CT CHEST",
    },
  },
});

const updateMetadata = {
  DICOMUpdates: {
    removableAttributes: new TextEncoder().encode(remove_attribute),
  },
};

await updateImageSetMetadata(
  datastoreID,
  imageSetID,
  versionID,
  updateMetadata,
);

```

```
);
```

사용 사례 3: 인스턴스를 제거합니다.

```
const remove_instance = JSON.stringify({
  SchemaVersion: 1.1,
  Study: {
    Series: {
      "1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
        Instances: {
          "1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {},
        },
      },
    },
  },
});

const updateMetadata = {
  DICOMUpdates: {
    removableAttributes: new TextEncoder().encode(remove_instance),
  },
};

await updateImageSetMetadata(
  datastoreID,
  imageSetID,
  versionID,
  updateMetadata,
);
```

사용 사례 4: 이전 버전으로 되돌립니다.

```
const updateMetadata = {
  revertToVersionId: "1",
};

await updateImageSetMetadata(
  datastoreID,
  imageSetID,
  versionID,
  updateMetadata,
```

```
);
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateImageSetMetadata](#)를 참조하세요.

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

시나리오

이미지 세트 및 이미지 프레임 시작하기

다음 코드 예제에서는 HealthImaging에서 DICOM 파일을 가져오고 이미지 프레임을 다운로드하는 방법을 보여줍니다.

구현은 명령줄 애플리케이션으로 구성됩니다.

- DICOM 가져오기의 리소스를 설정합니다.
- 데이터 스토어로 DICOM 파일을 가져옵니다.
- 가져오기 작업의 이미지 세트 ID를 검색합니다.
- 이미지 세트의 이미지 프레임 ID를 검색합니다.
- 이미지 프레임을 다운로드, 디코딩 및 확인합니다.
- 리소스를 정리합니다.

SDK for JavaScript(v3)

단계를 오케스트레이션합니다(index.js).

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
```



```
    loadState,
  } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "../deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "../dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "../import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "../image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "../image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "../verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "../clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
```

```
"Deploy Resources",
[
  deployStack,
  getStackName,
  getDatastoreName,
  getAccountId,
  createStack,
  waitForStackCreation,
  outputState,
  saveState,
],
context,
),
demo: new Scenario(
  "Run Demo",
  [
    loadState,
    doCopy,
    selectDataset,
    copyDataset,
    outputCopiedObjects,
    doImport,
    startDICOMImport,
    waitForImportJobCompletion,
    outputImportJobStatus,
    getManifestFile,
    parseManifestFile,
    outputImageSetIds,
    getImageSetMetadata,
    outputImageFrameIds,
    doVerify,
    decodeAndVerifyImages,
    saveState,
  ],
  context,
),
destroy: new Scenario(
  "Clean Up Resources",
  [loadState, confirmCleanup, deleteImageSets, deleteStack],
  context,
),
};

// Call function if run directly
```

```
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Health Imaging Workflow",
    description:
      "Work with DICOM images using an AWS Health Imaging data store.",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
  });
}
```

리소스를 배포합니다(deploy-steps.js).

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../scenarios/features/healthimaging_image_sets/resources/cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
```

```

    "deployStack",
    "Do you want to deploy the CloudFormation stack?",
    { type: "confirm" },
  );

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,

```

```

    },
    {
      ParameterKey: "userAccountID",
      ParameterValue: accountId,
    },
  ],
});

const response = await cfnClient.send(command);
state.stackId = response.StackId;
},
{ skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName === state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {} */ state) => !state.deployStack,
  },
);

```

```

);

export const outputState = new ScenarioOutput(
  "outputState",
  (** @type {} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
`;
  },
  { skipWhen: (** @type {} */ state) => !state.deployStack },
);

```

DICOM 파일을 복사합니다(dataset-steps.js).

```

import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",

```

```

        value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
      },
      {
        name: "MRI of head (192 images)",
        value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
      },
      {
        name: "MRI of breast (92 images)",
        value: "0002dd07-0b7f-4a68-a655-44461ca34096",
      },
    ],
  ];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",

```

```

async (/** @type { State } */ state) => {
  const inputBucket = state.stackOutputs.BucketName;
  const inputPrefix = "input/";
  const selectedDatasetId = state.selectDataset;

  const sourceBucket = "idc-open-data";
  const sourcePrefix = `${selectedDatasetId}`;

  const listObjectsCommand = new ListObjectsV2Command({
    Bucket: sourceBucket,
    Prefix: sourcePrefix,
  });

  const objects = await s3Client.send(listObjectsCommand);

  const copyPromises = objects.Contents.map((object) => {
    const sourceKey = object.Key;
    const destinationKey = `${inputPrefix}${sourceKey}
      .split("/")
      .slice(1)
      .join("/")}`;

    const copyCommand = new CopyObjectCommand({
      Bucket: inputBucket,
      CopySource: `/${sourceBucket}/${sourceKey}`,
      Key: destinationKey,
    });

    return s3Client.send(copyCommand);
  });

  const results = await Promise.all(copyPromises);
  state.copiedObjects = results.length;
},
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);

```

데이터 저장소로 가져오기를 시작합니다(import-steps.js).


```

import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,

```

```

        inputS3Uri,
        outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreId,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);

```

이미지 세트 ID를 가져옵니다(image-set-steps.js).

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

```

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce((ids, next) => {

```

```

        return Object.assign({}, ids, {
            [next.imageSetId]: next.imageSetId,
        });
    }, {}));
    state.imageSetIds = Object.keys(imageSetIds);
},
);

export const outputImageSetIds = new ScenarioOutput(
    "outputImageSetIds",
    (/** @type {State} */ state) =>
        `The image sets created by this import job are: \n${state.imageSetIds
            .map((id) => `Image set: ${id}`)
            .join("\n")}`
);

```

이미지 프레임 ID를 가져옵니다(image-frame-steps.js).

```

import {
    MedicalImagingClient,
    GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node:util";

import {
    ScenarioAction,
    ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID

```

```

    * @property {Array<{ Checksum: number, Height: number, Width: number }>}
    PixelDataChecksumFromBaseToFullResolution
    * @property {number} MinPixelValue
    * @property {number} MaxPixelValue
    * @property {number} FrameSizeInBytes
    */

/**
    * @typedef {Object} DICOMMetadata
    * @property {Object} DICOM
    * @property {DICOMValueRepresentation[]} DICOMVRs
    * @property {ImageFrameInformation[]} ImageFrames
    */

/**
    * @typedef {Object} Series
    * @property {{ [key: string]: DICOMMetadata }} Instances
    */

/**
    * @typedef {Object} Study
    * @property {Object} DICOM
    * @property {Series[]} Series
    */

/**
    * @typedef {Object} Patient
    * @property {Object} DICOM
    */

/**
    * @typedef {{
    *   SchemaVersion: string,
    *   DatastoreID: string,
    *   ImageSetID: string,
    *   Patient: Patient,
    *   Study: Study
    * }} ImageSetMetadata
    */

/**
    * @typedef {{ stackOutputs: {
    *   BucketName: string,
    *   DatastoreID: string,

```

```

*   RoleArn: string
* }, imageSetIds: string[] ]] State
*/

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreId,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>

```

```

        instance.ImageFrames.map((frame) => frame.ID),
    );

    output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
    "\n",
    )}\n\n`;
}

return output;
},
);

```

이미지 프레임을 확인합니다(verify-steps.js). [AWS HealthImaging Pixel Data Verification](#) 라이브러리가 확인에 사용되었습니다.

```

import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata

```

```

    * @property {Object} DICOM
    * @property {DICOMValueRepresentation[]} DICOMVRs
    * @property {ImageFrameInformation[]} ImageFrames
    */

/**
    * @typedef {Object} Series
    * @property {{ [key: string]: DICOMMetadata }} Instances
    */

/**
    * @typedef {Object} Study
    * @property {Object} DICOM
    * @property {Series[]} Series
    */

/**
    * @typedef {Object} Patient
    * @property {Object} DICOM
    */

/**
    * @typedef {{
    *   SchemaVersion: string,
    *   DatastoreID: string,
    *   ImageSetID: string,
    *   Patient: Patient,
    *   Study: Study
    * }} ImageSetMetadata
    */

/**
    * @typedef {{ stackOutputs: {
    *   BucketName: string,
    *   DatastoreID: string,
    *   RoleArn: string
    * }, imageSetMetadata: ImageSetMetadata[] }} State
    */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
  }

```



```

    default: true,
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreId;
      const imageSetId = metadata.ImageSetId;

      for (const [seriesInstanceId, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceId, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceId} and
sop ${sopInstanceId}`,
          );
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
              seriesInstanceId,
              sopInstanceId,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
              } else {
                reject(
                  new Error(

```

```

        `Verification tool exited with code ${code} for image set
        ${imageSetId}`,
        ),
    );
    }
    });
    });
    }
    }
    },
);

```

리소스를 폐기합니다(clean-up-steps.js).

```

import {
    CloudFormationClient,
    DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
    MedicalImagingClient,
    DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
    ScenarioAction,
    ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
    PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue

```

```

* @property {number} MaxPixelValue
* @property {number} FrameSizeInBytes
*/

/**
* @typedef {Object} DICOMMetadata
* @property {Object} DICOM
* @property {DICOMValueRepresentation[]} DICOMVRs
* @property {ImageFrameInformation[]} ImageFrames
*/

/**
* @typedef {Object} Series
* @property {{ [key: string]: DICOMMetadata }} Instances
*/

/**
* @typedef {Object} Study
* @property {Object} DICOM
* @property {Series[]} Series
*/

/**
* @typedef {Object} Patient
* @property {Object} DICOM
*/

/**
* @typedef {{
*   SchemaVersion: string,
*   DatastoreID: string,
*   ImageSetID: string,
*   Patient: Patient,
*   Study: Study
* }} ImageSetMetadata
*/

/**
* @typedef {{ stackOutputs: {
*   BucketName: string,
*   DatastoreID: string,
*   RoleArn: string
* }, imageSetMetadata: ImageSetMetadata[] }} State
*/

```

```

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreId;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({

```

```

    StackName: stackName,
  });

  await cfnClient.send(command);
  console.log(`Stack ${stackName} deletion initiated`);
},
{
  skipWhen: (/** @type {} */ state) => !state.confirmCleanup,
},
);

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.

- [DeleteImageSet](#)
- [GetDICOMImportJob](#)
- [GetImageFrame](#)
- [GetImageSetMetadata](#)
- [SearchImageSets](#)
- [StartDICOMImportJob](#)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

데이터 저장소에 태그 지정

다음 코드 예제에서는 HealthImaging 데이터 스토어에 태그를 지정하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이미지 세트 태그 지정

다음 코드 예제에서는 HealthImaging 이미지 세트에 태그를 지정하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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/";
  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

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK for JavaScript (v3)를 사용한 IAM 예제

다음 코드 예제에서는 IAM과 함께 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)
- [시나리오](#)

시작하기

Hello IAM

다음 코드 예제에서는 IAM 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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) {
      for (const policy of page.Policies) {
        console.log(`${policy.PolicyName}`);
        policyCount++;
      }
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListPolicies](#)를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 사용자를 생성하고 역할을 수입하는 방법을 보여줍니다.

Warning

보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마세요. 대신 [AWS IAM Identity Center](#)과 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 수입할 수 있도록 정책을 추가합니다.
- 역할을 수입하고 임시 자격 증명 정보를 사용하여 S3 버킷을 나열한 후 리소스를 정리합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon S3 버킷을 나열할 수 있는 권한을 부여하는 역할과 IAM 사용자를 생성합니다. 사용자는 역할을 수입할 수 있는 권한만 있습니다. 역할을 수입한 후 임시 자격 증명을 사용하여 계정의 버킷을 나열합니다.

```
import {
  CreateUserCommand,
  GetUserCommand,
  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";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "iam_basic_test_username";
const policyName = "iam_basic_test_policy";
const roleName = "iam_basic_test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
  } catch (caught) {

```



```

    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
      return User;
    }
    throw caught;
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  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: 120 }, () =>
  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](#)

작업

AttachRolePolicy

다음 코드 예시는 AttachRolePolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [AttachRolePolicy](#)를 참조하세요.

CreateAccessKey

다음 코드 예시는 CreateAccessKey의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateAccessKey](#)를 참조하세요.

CreateAccountAlias

다음 코드 예시는 CreateAccountAlias의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateAccountAlias](#)를 참조하세요.

CreateGroup

다음 코드 예시는 CreateGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateGroup](#)을 참조하세요.

CreateInstanceProfile

다음 코드 예시는 CreateInstanceProfile의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateInstanceProfile](#)을 참조하세요.

CreatePolicy

다음 코드 예시는 CreatePolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreatePolicy](#)를 참조하세요.

CreateRole

다음 코드 예시는 CreateRole의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조](#)의 CreateRole을 참조하세요.

CreateSAMLProvider

다음 코드 예시는 CreateSAMLProvider의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import * as path from "node: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 [CreateSAMLProvider](#)를 참조하세요.

CreateServiceLinkedRole

다음 코드 예시는 CreateServiceLinkedRole의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

서비스 연결 역할을 생성합니다.

```

import {
  CreateServiceLinkedRoleCommand,

```

```

    GetRoleCommand,
    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,
    });
    try {
      const response = await client.send(command);
      console.log(response);
      return response;
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidInputException" &&
        caught.message.includes(
          "Service role name AWSServiceRoleForElasticBeanstalk has been taken in this
account",
        )
      ) {
        console.warn(caught.message);
        return client.send(
          new GetRoleCommand({ RoleName: "AWSServiceRoleForElasticBeanstalk" }),
        );
      }
      throw caught;
    }
  };

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateServiceLinkedRole](#)을 참조하세요.

CreateUser

다음 코드 예시는 CreateUser의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 [AWS SDK for JavaScript API 참조](#)의 CreateUser를 참조하세요.

DeleteAccessKey

다음 코드 예시는 DeleteAccessKey의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteAccessKey](#)를 참조하세요.

DeleteAccountAlias

다음 코드 예시는 DeleteAccountAlias의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteAccountAlias](#)를 참조하세요.

DeleteGroup

다음 코드 예시는 DeleteGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteGroup](#)을 참조하세요.

DeleteInstanceProfile

다음 코드 예시는 DeleteInstanceProfile의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteInstanceProfile](#)을 참조하세요.

DeletePolicy

다음 코드 예시는 DeletePolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeletePolicy](#)를 참조하세요.

DeleteRole

다음 코드 예시는 DeleteRole의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteRole](#)을 참조하세요.

DeleteRolePolicy

다음 코드 예시는 DeleteRolePolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteRolePolicy](#)를 참조하세요.

DeleteSAMLProvider

다음 코드 예시는 DeleteSAMLProvider의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 [DeleteSAMLProvider](#)를 참조하세요.

DeleteServerCertificate

다음 코드 예시는 DeleteServerCertificate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteServerCertificate](#)를 참조하세요.

DeleteServiceLinkedRole

다음 코드 예시는 DeleteServiceLinkedRole의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteServiceLinkedRole](#)을 참조하세요.

DeleteUser

다음 코드 예시는 DeleteUser의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteUser](#)를 참조하세요.

DetachRolePolicy

다음 코드 예시는 DetachRolePolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DetachRolePolicy](#)를 참조하세요.

GetAccessKeyLastUsed

다음 코드 예시는 GetAccessKeyLastUsed의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetAccessKeyLastUsed](#)를 참조하세요.

GetAccountPasswordPolicy

다음 코드 예시는 GetAccountPasswordPolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetAccountPasswordPolicy](#)를 참조하세요.

GetPolicy

다음 코드 예시는 GetPolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetPolicy](#)를 참조하세요.

GetRole

다음 코드 예시는 GetRole의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetRole](#)을 참조하세요.

GetServerCertificate

다음 코드 예시는 GetServerCertificate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetServerCertificate](#)를 참조하세요.

GetServiceLinkedRoleDeletionStatus

다음 코드 예시는 GetServiceLinkedRoleDeletionStatus의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetServiceLinkedRoleDeletionStatus](#)를 참조하세요.

ListAccessKeys

다음 코드 예시는 ListAccessKeys의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListAccessKeys](#)를 참조하세요.

ListAccountAliases

다음 코드 예시는 ListAccountAliases의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListAccountAliases](#)를 참조하세요.

ListAttachedRolePolicies

다음 코드 예시는 ListAttachedRolePolicies의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListAttachedRolePolicies](#)를 참조하세요.

ListGroups

다음 코드 예시는 ListGroups의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListGroups](#)를 참조하세요.

ListPolicies

다음 코드 예시는 ListPolicies의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListPolicies](#)를 참조하세요.

ListRolePolicies

다음 코드 예시는 ListRolePolicies의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListRolePolicies](#)를 참조하세요.

ListRoles

다음 코드 예시는 ListRoles의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListRoles](#)를 참조하세요.

ListSAMLProviders

다음 코드 예시는 ListSAMLProviders의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 [ListSAMLProviders](#)를 참조하세요.

ListServerCertificates

다음 코드 예시는 ListServerCertificates의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListServerCertificates](#)를 참조하세요.

ListUsers

다음 코드 예시는 ListUsers의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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);

  for (const { UserName, CreateDate } of response.Users) {
    console.log(`${UserName} created on: ${CreateDate}`);
  }
  return response;
};
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListUsers](#)를 참조하세요.

PutRolePolicy

다음 코드 예시는 PutRolePolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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:::amzn-s3-demo-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 참조의 [PutRolePolicy](#)를 참조하세요.

UpdateAccessKey

다음 코드 예시는 UpdateAccessKey의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateAccessKey](#)를 참조하세요.

UpdateServerCertificate

다음 코드 예시는 UpdateServerCertificate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateServerCertificate](#)를 참조하세요.

UpdateUser

다음 코드 예시는 UpdateUser의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateUser](#)를 참조하세요.

UploadServerCertificate

다음 코드 예시는 UploadServerCertificate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "node: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 참조의 [UploadServerCertificate](#)를 참조하세요.

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.
- Auto Scaling 그룹의 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 파라미터를 업데이트하여 요청 및 상태 확인에 대한 웹 서버 응답을 제어합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배우보세요.

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 "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}

```

모든 리소스를 배포하기 위한 단계를 생성합니다.

```

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 { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 () => {
    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,
            },
        }),
    );
}),
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) => {
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  ),
});

```

```

    })),
  );
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    })
  );
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  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,
    )),
);
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) => {
    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] },
    );
}),
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) => {
    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 },
        ],
      }),
    );
    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) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  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),
        );
      }
      return MESSAGES.noIpRules;
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      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("${IP_ADDRESS}", state.myIp);
    }
    return false;
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${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,
      );
    }
  }),
  saveState,
];

```

데모를 실행하기 위한 단계를 생성합니다.

```

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) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  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: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      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];
        const ec2Client = new EC2Client({});
        const { IamInstanceProfileAssociations } = await ec2Client.send(
            new DescribeIamInstanceProfileAssociationsCommand({
                Filters: [
                    { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
                ],
            }),
        );
        state.instanceProfileAssociationId =
            IamInstanceProfileAssociations[0].AssociationId;
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            ec2Client.send(
                new ReplaceIamInstanceProfileAssociationCommand({
                    AssociationId: state.instanceProfileAssociationId,
                    IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
                }),
            ),
        );

        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",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
```

```

        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    })),
);
await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
    new AddRoleToInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
    })),
);

return InstanceProfile;
}

```

모든 리소스를 폐기하는 단계를 생성합니다.

```

import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
    EC2Client,
    DeleteKeyPairCommand,
    DeleteLaunchTemplateCommand,
    RevokeSecurityGroupIngressCommand,
} 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 { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 = [
  loadState,
  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,
      );
    }
  })
];

```

```

    );
  }
  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(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }
  return MESSAGES.deletedKeyPair.replace(
    "${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);
  }
  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,
    );
  }
  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);
      }
      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,
        );
      }
    })
  ],

```

```

    }
    return MESSAGES.deletedInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }},
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }},
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
    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(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }
  return MESSAGES.deletedAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  );
}),
new ScenarioAction("deleteLaunchTemplate", async (state) => {
  const client = new EC2Client({});
  try {
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
  } catch (e) {
    state.deleteLaunchTemplateError = e;
  }
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
  if (state.deleteLaunchTemplateError) {
    console.error(state.deleteLaunchTemplateError);
    return MESSAGES.deleteLaunchTemplateError.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
  return MESSAGES.deletedLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  );
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
    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.");
      }
    });
  } catch (e) {
    state.deleteLoadBalancerError = e;
  }
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
  return MESSAGES.deletedLoadBalancer.replace(
    "${LB_NAME}",
    NAMES.loadBalancerName,
  );
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  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;
  }
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {

```

```

    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${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);
    }
    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);
  }
  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");
  }
  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(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  }
  return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
    "${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,
    );
  }
  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,
        );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
    );
}),
new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
        /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
        state,
    ) => {
        const ec2Client = new EC2Client({});

        try {
            await ec2Client.send(
                new RevokeSecurityGroupIngressCommand({
                    GroupId: state.defaultSecurityGroup.GroupId,
                    CidrIp: `${state.myIp}/32`,
                    FromPort: 80,
                    ToPort: 80,
                })
            );
        }
    }
),

```

```

        IpProtocol: "tcp",
      )),
    );
  } catch (e) {
    state.revokeSecurityGroupIngressError = e;
  }
},
),
new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
  if (state.revokeSecurityGroupIngressError) {
    console.error(state.revokeSecurityGroupIngressError);
    return MESSAGES.revokeSecurityGroupIngressError.replace(
      "${IP}",
      state.myIp,
    );
  }
  return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
}),
];

/**
 * @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;
    }
    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](#)

AWS IoT SiteWise SDK for JavaScript(v3)를 사용한 예제

다음 코드 예제에서는 AWS SDK for JavaScript (v3)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS IoT SiteWise.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)

시작하기

안녕하세요 AWS IoT SiteWise

다음 코드 예제에서는 AWS IoT SiteWise를 사용하여 시작하는 방법을 보여 줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  paginateListAssetModels,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";

// Call ListDocuments and display the result.
```

```
export const main = async () => {
  const client = new IoTSiteWiseClient();
  const listAssetModelsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListAssetModels({ client }, { maxResults: 5 });
    for await (const page of paginator) {
      listAssetModelsPaginated.push(...page.assetModelSummaries);
    }
  } catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
  }
  for (const { name, creationDate } of listAssetModelsPaginated) {
    console.log(`${name} - ${creationDate}`);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListAssetModels](#)을 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- AWS IoT SiteWise 자산 모델을 생성합니다.
- AWS IoT SiteWise 자산을 생성합니다.
- 속성 ID 값을 검색합니다.
- AWS IoT SiteWise 애셋으로 데이터를 전송합니다.
- AWS IoT SiteWise Asset 속성의 값을 검색합니다.

- AWS IoT SiteWise 포털을 생성합니다.
- AWS IoT SiteWise 게이트웨이를 생성합니다.
- AWS IoT SiteWise 게이트웨이를 설명합니다.
- AWS IoT SiteWise 자산을 삭제합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
  //} from "@aws-doc-sdk-examples/lib/scenario/index.js";
} from "../../libs/scenario/index.js";
import {
  IoTSiteWiseClient,
  CreateAssetModelCommand,
  CreateAssetCommand,
  ListAssetModelPropertiesCommand,
  BatchPutAssetPropertyValueCommand,
  GetAssetPropertyValueCommand,
  CreatePortalCommand,
  DescribePortalCommand,
  CreateGatewayCommand,
  DescribeGatewayCommand,
  DeletePortalCommand,
  DeleteGatewayCommand,
  DeleteAssetCommand,
  DeleteAssetModelCommand,
  DescribeAssetModelCommand,
} from "@aws-sdk/client-iotsitewise";
import {
  CloudFormationClient,
  CreateStackCommand,
```

```

    DeleteStackCommand,
    DescribeStacksCommand,
    waitUntilStackExists,
    waitUntilStackCreateComplete,
    waitUntilStackDeleteComplete,
  } from "@aws-sdk/client-cloudformation";
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { parseArgs } from "node:util";
import { readFileSync } from "node:fs";
import { fileURLToPath } from "node:url";
import { dirname } from "node:path";

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);
const stackName = "SiteWiseBasicsStack";

/**
 * @typedef {{
 *   iotSiteWiseClient: import('@aws-sdk/client-iotsitewise').IotSiteWiseClient,
 *   cloudFormationClient: import('@aws-sdk/client-
cloudformation').CloudFormationClient,
 *   stackName,
 *   stack,
 *   askToDeleteResources: true,
 *   asset: {assetName: "MyAsset1"},
 *   assetModel: {assetModelName: "MyAssetModel1"},
 *   portal: {portalName: "MyPortal1"},
 *   gateway: {gatewayName: "MyGateway1"},
 *   propertyIds: [],
 *   contactEmail: "user@mydomain.com",
 *   thing: "MyThing1",
 *   sampleData: { temperature: 23.5, humidity: 65.0}
 * }} State
 */

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
});

const greet = new ScenarioOutput(

```

```

    "greet",
    `AWS IoT SiteWise is a fully managed industrial software-as-a-service (SaaS)
    that makes it easy to collect, store, organize, and monitor data from industrial
    equipment and processes. It is designed to help industrial and manufacturing
    organizations collect data from their equipment and processes, and use that data to
    make informed decisions about their operations.
    One of the key features of AWS IoT SiteWise is its ability to connect to a wide
    range of industrial equipment and systems, including programmable logic controllers
    (PLCs), sensors, and other industrial devices. It can collect data from these
    devices and organize it into a unified data model, making it easier to analyze and
    gain insights from the data. AWS IoT SiteWise also provides tools for visualizing
    the data, setting up alarms and alerts, and generating reports.
    Another key feature of AWS IoT SiteWise is its ability to scale to handle large
    volumes of data. It can collect and store data from thousands of devices and
    process millions of data points per second, making it suitable for large-scale
    industrial operations. Additionally, AWS IoT SiteWise is designed to be secure
    and compliant, with features like role-based access controls, data encryption,
    and integration with other AWS services for additional security and compliance
    features.

    Let's get started...`,
    { header: true },
  );

  const displayBuildCloudFormationStack = new ScenarioOutput(
    "displayBuildCloudFormationStack",
    "This scenario uses AWS CloudFormation to create an IAM role that is required for
    this scenario. The stack will now be deployed.",
  );

  const sdkBuildCloudFormationStack = new ScenarioAction(
    "sdkBuildCloudFormationStack",
    async (** @type {State} */ state) => {
      try {
        const data = readFileSync(
          `${__dirname}/../../../../../resources/cfn/iotsitewise_basics/SitewiseRoles-
          template.yml`,
          "utf8",
        );
        await state.cloudFormationClient.send(
          new CreateStackCommand({
            StackName: stackName,
            TemplateBody: data,
            Capabilities: ["CAPABILITY_IAM"],

```

```

    }},
  );
  await waitUntilStackExists(
    { client: state.cloudFormationClient },
    { StackName: stackName },
  );
  await waitUntilStackCreateComplete(
    { client: state.cloudFormationClient },
    { StackName: stackName },
  );
  const stack = await state.cloudFormationClient.send(
    new DescribeStacksCommand({
      StackName: stackName,
    }),
  );
  state.stack = stack.Stacks[0].Outputs[0];
  console.log(`The ARN of the IAM role is ${state.stack.OutputValue}`);
} catch (caught) {
  console.error(caught.message);
  throw caught;
}
},
);

```

```

const displayCreateAWSSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSSiteWiseAssetModel",
  `1. Create an AWS SiteWise Asset Model

```

An AWS IoT SiteWise Asset Model is a way to represent the physical assets, such as equipment, processes, and systems, that exist in an industrial environment. This model provides a structured and hierarchical representation of these assets, allowing users to define the relationships and properties of each asset.

```

This scenario creates two asset model properties: temperature and humidity.`
);

```

```

const sdkCreateAWSSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSSiteWiseAssetModel",
  async (** @type {State} */ state) => {
    let assetModelResponse;
    try {
      assetModelResponse = await state.iotSiteWiseClient.send(
        new CreateAssetModelCommand({
          assetModelName: state.assetModel.assetModelName,
          assetModelProperties: [

```

```

        {
            name: "Temperature",
            dataType: "DOUBLE",
            type: {
                measurement: {},
            },
        },
        {
            name: "Humidity",
            dataType: "DOUBLE",
            type: {
                measurement: {},
            },
        },
    ],
    )),
);
state.assetModel.assetModelId = assetModelResponse.assetModelId;
console.log(
    `Asset Model successfully created. Asset Model ID:
    ${state.assetModel.assetModelId}`,
);
} catch (caught) {
    if (caught.name === "ResourceAlreadyExistsException") {
        console.log(
            `The Asset Model ${state.assetModel.assetModelName} already exists.`,
        );
        throw caught;
    }
    console.error(`${caught.message}`);
    throw caught;
}
},
);

const displayCreateAWSIoTSiteWiseAssetModel = new ScenarioOutput(
    "displayCreateAWSIoTSiteWiseAssetModel",
    `2. Create an AWS IoT SiteWise Asset
    The IoT SiteWise model that we just created defines the structure and metadata for
    your physical assets. Now we create an asset from the asset model.

    Let's wait 30 seconds for the asset to be ready.`,
);

```



```

const waitThirtySeconds = new ScenarioAction("waitThirtySeconds", async () => {
  await wait(30); // wait 30 seconds
  console.log("Time's up! Let's check the asset's status.");
});

const sdkCreateAWSIoTSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSIoTSiteWiseAssetModel",
  async (** @type {State} */ state) => {
    try {
      const assetResponse = await state.iotSiteWiseClient.send(
        new CreateAssetCommand({
          assetModelId: state.assetModel.assetModelId,
          assetName: state.asset.assetName,
        })),
      );
      state.asset.assetId = assetResponse.assetId;
      console.log(`Asset created with ID: ${state.asset.assetId}`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(
          `The Asset ${state.assetModel.assetModelName} was not found.`
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayRetrievePropertyId = new ScenarioOutput(
  "displayRetrievePropertyId",
  `3. Retrieve the property ID values

```

To send data to an asset, we need to get the property ID values. In this scenario, we access the temperature and humidity property ID values.`

```

);

```

```

const sdkRetrievePropertyId = new ScenarioAction(
  "sdkRetrievePropertyId",
  async (state) => {
    try {
      const retrieveResponse = await state.iotSiteWiseClient.send(
        new ListAssetModelPropertiesCommand({

```

```

        assetModelId: state.assetModel.assetModelId,
    )),
    );
    for (const retrieveResponseKey in
retrieveResponse.assetModelPropertySummaries) {
        if (
            retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
                .name === "Humidity"
        ) {
            state.propertyIds.Humidity =
                retrieveResponse.assetModelPropertySummaries[
                    retrieveResponseKey
                ].id;
        }
        if (
            retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
                .name === "Temperature"
        ) {
            state.propertyIds.Temperature =
                retrieveResponse.assetModelPropertySummaries[
                    retrieveResponseKey
                ].id;
        }
    }
    console.log(`The Humidity propertyId is ${state.propertyIds.Humidity}`);
    console.log(
        `The Temperature propertyId is ${state.propertyIds.Temperature}`,
    );
} catch (caught) {
    if (caught.name === "IoTSiteWiseException") {
        console.log(
            `There was a problem retrieving the properties: ${caught.message}`,
        );
        throw caught;
    }
    console.error(`${caught.message}`);
    throw caught;
}
    },
);

const displaySendDataToIoTSiteWiseAsset = new ScenarioOutput(
    "displaySendDataToIoTSiteWiseAsset",
    `4. Send data to an AWS IoT SiteWise Asset

```

By sending data to an IoT SiteWise Asset, you can aggregate data from multiple sources, normalize the data into a standard format, and store it in a centralized location. This makes it easier to analyze and gain insights from the data.

In this example, we generate sample temperature and humidity data and send it to the AWS IoT SiteWise asset.`

```
);
```

```
const sdkSendDataToIoTSiteWiseAsset = new ScenarioAction(
  "sdkSendDataToIoTSiteWiseAsset",
  async (state) => {
    try {
      const sendResponse = await state.iotSiteWiseClient.send(
        new BatchPutAssetPropertyValueCommand({
          entries: [
            {
              entryId: "entry-3",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Humidity,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.humidity,
                  },
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
            {
              entryId: "entry-4",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Temperature,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.temperature,
                  },
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
          ],
        })
      );
    } catch (error) {
      console.error(error);
    }
  }
);
```

```

        ],
      },
    ],
  )),
);
console.log("The data was sent successfully.");
} catch (caught) {
  if (caught.name === "ResourceNotFoundException") {
    console.log(`The Asset ${state.asset.assetName} was not found.`);
    throw caught;
  }
  console.error(`${caught.message}`);
  throw caught;
}
},
);

```

```

const displayRetrieveValueOfIoTSiteWiseAsset = new ScenarioOutput(
  "displayRetrieveValueOfIoTSiteWiseAsset",
  `5. Retrieve the value of the IoT SiteWise Asset property

```

IoT SiteWise is an AWS service that allows you to collect, process, and analyze industrial data from connected equipment and sensors. One of the key benefits of reading an IoT SiteWise property is the ability to gain valuable insights from your industrial data.`,

```

);

```

```

const sdkRetrieveValueOfIoTSiteWiseAsset = new ScenarioAction(
  "sdkRetrieveValueOfIoTSiteWiseAsset",
  async (** @type {State} */ state) => {
    try {
      const temperatureResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
          propertyId: state.propertyIds.Temperature,
        }),
      );
      const humidityResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
          propertyId: state.propertyIds.Humidity,
        }),
      );
      console.log(

```

```

        `The property value for Temperature is
        ${temperatureResponse.propertyValue.value.doubleValue}`,
    );
    console.log(
        `The property value for Humidity is
        ${humidityResponse.propertyValue.value.doubleValue}`,
    );
} catch (caught) {
    if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
        throw caught;
    }
    console.error(`${caught.message}`);
    throw caught;
}
},
);

```

```

const displayCreateIoTSiteWisePortal = new ScenarioOutput(
    "displayCreateIoTSiteWisePortal",
    `6. Create an IoT SiteWise Portal

```

An IoT SiteWise Portal allows you to aggregate data from multiple industrial sources, such as sensors, equipment, and control systems, into a centralized platform.`,

```

);

```

```

const sdkCreateIoTSiteWisePortal = new ScenarioAction(
    "sdkCreateIoTSiteWisePortal",
    async (** @type {State} */ state) => {
        try {
            const createPortalResponse = await state.iotSiteWiseClient.send(
                new CreatePortalCommand({
                    portalName: state.portal.portalName,
                    portalContactEmail: state.contactEmail,
                    roleArn: state.stack.OutputValue,
                }),
            );
            state.portal = { ...state.portal, ...createPortalResponse };
            await wait(5); // Allow the portal to properly propagate.
            console.log(
                `Portal created successfully. Portal ID ${createPortalResponse.portalId}`,
            );
        } catch (caught) {

```

```

        if (caught.name === "IoTSiteWiseException") {
            console.log(
                `There was a problem creating the Portal: ${caught.message}.`,
            );
            throw caught;
        }
        console.error(`${caught.message}`);
        throw caught;
    }
},
);

const displayDescribePortal = new ScenarioOutput(
    "displayDescribePortal",
    `7. Describe the Portal

In this step, we get a description of the portal and display the portal URL.`,
);

const sdkDescribePortal = new ScenarioAction(
    "sdkDescribePortal",
    async (** @type {State} */ state) => {
        try {
            const describePortalResponse = await state.iotSiteWiseClient.send(
                new DescribePortalCommand({
                    portalId: state.portal.portalId,
                }),
            );
            console.log(`Portal URL: ${describePortalResponse.portalStartUrl}`);
        } catch (caught) {
            if (caught.name === "ResourceNotFoundException") {
                console.log(`The Portal ${state.portal.portalName} was not found.`);
                throw caught;
            }
            console.error(`${caught.message}`);
            throw caught;
        }
    },
);

const displayCreateIoTSiteWiseGateway = new ScenarioOutput(
    "displayCreateIoTSiteWiseGateway",
    `8. Create an IoT SiteWise Gateway

```

```

IoT SiteWise Gateway serves as the bridge between industrial equipment, sensors, and
the cloud-based IoT SiteWise service. It is responsible for securely collecting,
processing, and transmitting data from various industrial assets to the IoT
SiteWise platform, enabling real-time monitoring, analysis, and optimization of
industrial operations.`
);

const sdkCreateIoTSiteWiseGateway = new ScenarioAction(
  "sdkCreateIoTSiteWiseGateway",
  async (** @type {State} */ state) => {
    try {
      const createGatewayResponse = await state.iotSiteWiseClient.send(
        new CreateGatewayCommand({
          gatewayName: state.gateway.gatewayName,
          gatewayPlatform: {
            greengrassV2: {
              coreDeviceThingName: state.thing,
            },
          },
        }),
      );
      console.log(
        `Gateway creation completed successfully. ID is
        ${createGatewayResponse.gatewayId}`,
      );
      state.gateway.gatewayId = createGatewayResponse.gatewayId;
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem creating the gateway: ${caught.message}.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayDescribeIoTSiteWiseGateway = new ScenarioOutput(
  "displayDescribeIoTSiteWiseGateway",
  "9. Describe the IoT SiteWise Gateway",
);

```

```

const sdkDescribeIoTSiteWiseGateway = new ScenarioAction(
  "sdkDescribeIoTSiteWiseGateway",
  async (** @type {State} */ state) => {
    try {
      const describeGatewayResponse = await state.iotSiteWiseClient.send(
        new DescribeGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log("Gateway creation completed successfully.");
      console.log(`Gateway Name: ${describeGatewayResponse.gatewayName}`);
      console.log(`Gateway ARN: ${describeGatewayResponse.gatewayArn}`);
      console.log(
        `Gateway Platform: ${Object.keys(describeGatewayResponse.gatewayPlatform)}`,
      );
      console.log(
        `Gateway Creation Date: ${describeGatewayResponse.creationDate}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const askToDeleteResources = new ScenarioInput(
  "askToDeleteResources",
  `10. Delete the AWS IoT SiteWise Assets

Before you can delete the Asset Model, you must delete the assets.`
  , { type: "confirm" },
);

const displayConfirmDeleteResources = new ScenarioAction(
  "displayConfirmDeleteResources",
  async (** @type {State} */ state) => {
    if (state.askToDeleteResources) {
      return "You selected to delete the SiteWise assets.";
    }
  }
);

```



```
    return "The resources will not be deleted. Please delete them manually to avoid
    charges.";
  },
);

const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (** @type {State} */ state) => {
    await wait(10); // Give the portal status time to catch up.
    try {
      await state.iotSiteWiseClient.send(
        new DeletePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(
        `Portal ${state.portal.portalName} was deleted successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
      } else {
        console.log(`When trying to delete the portal: ${caught.message}`);
      }
    }

    try {
      await state.iotSiteWiseClient.send(
        new DeleteGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log(
        `Gateway ${state.gateway.gatewayName} was deleted successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
      } else {
        console.log(`When trying to delete the gateway: ${caught.message}`);
      }
    }

    try {
```

```
    await state.iotSiteWiseClient.send(
      new DeleteAssetCommand({
        assetId: state.asset.assetId,
      }),
    );
    await wait(5); // Allow the delete to finish.
    console.log(`Asset ${state.asset.assetName} was deleted successfully.`);
  } catch (caught) {
    if (caught.name === "ResourceNotFoundException") {
      console.log(`The Asset ${state.asset.assetName} was not found.`);
    } else {
      console.log(`When deleting the asset: ${caught.message}`);
    }
  }
}

await wait(30); // Allow asset deletion to finish.
try {
  await state.iotSiteWiseClient.send(
    new DeleteAssetModelCommand({
      assetModelId: state.assetModel.assetModelId,
    }),
  );
  console.log(
    `Asset Model ${state.assetModel.assetModelName} was deleted successfully.`,
  );
} catch (caught) {
  if (caught.name === "ResourceNotFoundException") {
    console.log(
      `The Asset Model ${state.assetModel.assetModelName} was not found.`,
    );
  } else {
    console.log(`When deleting the asset model: ${caught.message}`);
  }
}

try {
  await state.cloudFormationClient.send(
    new DeleteStackCommand({
      StackName: stackName,
    }),
  );
  await waitUntilStackDeleteComplete(
    { client: state.cloudFormationClient },
    { StackName: stackName },
  );
}
```

```

    );
    console.log("The stack was deleted successfully.");
  } catch (caught) {
    console.log(
      `${caught.message}. The stack was NOT deleted. Please clean up the resources manually.`
    );
  }
},
{ skipWhen: (/** @type {} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the IoT Sitewise Basics scenario for the AWS Javascript SDK v3. Thank you!",
);

const myScenario = new Scenario(
  "IoTSiteWise Basics",
  [
    greet,
    pressEnter,
    displayBuildCloudFormationStack,
    sdkBuildCloudFormationStack,
    pressEnter,
    displayCreateAWSSiteWiseAssetModel,
    sdkCreateAWSSiteWiseAssetModel,
    displayCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    waitThirtySeconds,
    sdkCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    displayRetrievePropertyId,
    sdkRetrievePropertyId,
    pressEnter,
    displaySendDataToIoTSiteWiseAsset,
    sdkSendDataToIoTSiteWiseAsset,
    pressEnter,
    displayRetrieveValueOfIoTSiteWiseAsset,
    sdkRetrieveValueOfIoTSiteWiseAsset,
    pressEnter,
    displayCreateIoTSiteWisePortal,
    sdkCreateIoTSiteWisePortal,
  ]
);

```

```

    pressEnter,
    displayDescribePortal,
    sdkDescribePortal,
    pressEnter,
    displayCreateIoTSiteWiseGateway,
    sdkCreateIoTSiteWiseGateway,
    pressEnter,
    displayDescribeIoTSiteWiseGateway,
    sdkDescribeIoTSiteWiseGateway,
    pressEnter,
    askToDeleteResources,
    displayConfirmDeleteResources,
    sdkDeleteResources,
    goodbye,
  ],
  {
    iotSiteWiseClient: new IoTSiteWiseClient({}),
    cloudFormationClient: new CloudFormationClient({}),
    asset: { assetName: "MyAsset1" },
    assetModel: { assetModelName: "MyAssetModel1" },
    portal: { portalName: "MyPortal1" },
    gateway: { gatewayName: "MyGateway1" },
    propertyIds: [],
    contactEmail: "user@mydomain.com",
    thing: "MyThing1",
    sampleData: { temperature: 23.5, humidity: 65.0 },
  },
);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  },
);
});

```

```
main({ confirmAll: values.yes });
}
```

작업

BatchPutAssetPropertyValue

다음 코드 예시는 BatchPutAssetPropertyValue의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  BatchPutAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Batch put asset property values.
 * @param {{ entries : array }}
 */
export const main = async ({ entries }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new BatchPutAssetPropertyValueCommand({
        entries: entries,
      }),
    );
    console.log("Asset properties batch put successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
```

```

        console.warn(`${caught.message}. A resource could not be found.`);
    } else {
        throw caught;
    }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [BatchPutAssetPropertyValue](#)를 참조하세요.

CreateAsset

다음 코드 예시는 CreateAsset의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import {
    CreateAssetCommand,
    IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an Asset.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetName, assetModelId }) => {
    const client = new IoTSiteWiseClient({});
    try {
        const result = await client.send(
            new CreateAssetCommand({
                assetName: assetName, // The name to give the Asset.
                assetModelId: assetModelId, // The ID of the asset model from which to
                create the asset.
            })
        );
    } catch (error) {
        console.error(error);
    }
};

```

```

    }},
  );
  console.log("Asset created successfully.");
  return result;
} catch (caught) {
  if (caught instanceof Error && caught.name === "ResourceNotFound") {
    console.warn(
      `${caught.message}. The asset model could not be found. Please check the
asset model id.`
    );
  } else {
    throw caught;
  }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateAsset](#)을 참조하세요.

CreateAssetModel

다음 코드 예시는 CreateAssetModel의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import {
  CreateAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an Asset Model.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetModelName, assetModelId }) => {

```

```
const client = new IoTSiteWiseClient({});
try {
  const result = await client.send(
    new CreateAssetModelCommand({
      assetModelName: assetModelName, // The name to give the Asset Model.
    }),
  );
  console.log("Asset model created successfully.");
  return result;
} catch (caught) {
  if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
    console.warn(
      `${caught.message}. There was a problem creating the asset model.`
    );
  } else {
    throw caught;
  }
}
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateAssetModel](#)을 참조하세요.

CreateGateway

다음 코드 예시는 CreateGateway의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  CreateGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
```



```

* Create a Gateway.
* @param {{ }}
*/
export const main = async ({ gatewayName }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateGatewayCommand({
        gatewayName: gatewayName, // The name to give the created Gateway.
      }),
    );
    console.log("Gateway created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the Gateway.`,
      );
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateGateway](#)를 참조하세요.

DeleteAsset

다음 코드 예시는 DeleteAsset의 사용 방법을 보여 줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import {
  DeleteAssetCommand,
  IoTSiteWiseClient,

```

```
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset.
 * @param {{ assetId : string }}
 */
export const main = async ({ assetId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetCommand({
        assetId: assetId, // The model id to delete.
      }),
    );
    console.log("Asset deleted successfully.");
    return { assetDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset.`
      );
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteAsset](#)을 참조하세요.

DeleteAssetModel

다음 코드 예시는 DeleteAssetModel의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  DeleteAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetModelCommand({
        assetModelId: assetModelId, // The model id to delete.
      }),
    );
    console.log("Asset model deleted successfully.");
    return { assetModelDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset model.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteAssetModel](#)을 참조하세요.

DeleteGateway

다음 코드 예시는 DeleteGateway의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  DeleteGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway deleted successfully.");
    return { gatewayDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Gateway could not be found. Please check the Gateway Id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteGateway](#)를 참조하세요.

DescribeAssetModel

다음 코드 예시는 DescribeAssetModel의 사용 방법을 보여 줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  DescribeAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const { assetModelDescription } = await client.send(
      new DescribeAssetModelCommand({
        assetModelId: assetModelId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Asset model information retrieved successfully.");
    return { assetModelDescription: assetModelDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset model could not be found. Please check the asset model id.`
      );
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeAssetModel](#)을 참조하세요.

DescribeGateway

다음 코드 예시는 DescribeGateway의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  DescribeGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const { gatewayDescription } = await client.send(
      new DescribeGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway information retrieved successfully.");
    return { gatewayDescription: gatewayDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Gateway could not be found. Please check the Gateway Id.`
      );
    }
  }
};
```

```

    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeGateway](#)를 참조하세요.

GetAssetPropertyValue

다음 코드 예시는 GetAssetPropertyValue의 사용 방법을 보여 줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import {
  GetAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset property value.
 * @param {{ entryId : string }}
 */
export const main = async ({ entryId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new GetAssetPropertyValueCommand({
        entryId: entryId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Asset property information retrieved successfully.");
    return result;
  } catch (caught) {

```

```

    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset property entry could not be found. Please
check the entry id.`
      );
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetAssetPropertyValue](#)를 참조하세요.

ListAssetModels

다음 코드 예시는 ListAssetModels의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import {
  ListAssetModelsCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * List asset models.
 * @param {{ assetModelTypes : array }}
 */
export const main = async ({ assetModelTypes = [] }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new ListAssetModelsCommand({
        assetModelTypes: assetModelTypes, // The model types to list

```



```

    }},
  );
  console.log("Asset model types retrieved successfully.");
  return result;
} catch (caught) {
  if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
    console.warn(
      `${caught.message}. There was a problem listing the asset model types.`
    );
  } else {
    throw caught;
  }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListAssetModels](#)을 참조하세요.

SDK for JavaScript(v3)를 사용한 Kinesis 예제

다음 코드 예제에서는 Kinesis에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)
- [서버리스 예제](#)

작업

PutRecords

다음 코드 예시는 PutRecords의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { PutRecordsCommand, KinesisClient } from "@aws-sdk/client-kinesis";

/**
 * Put multiple records into a Kinesis stream.
 * @param {{ streamArn: string }} config
 */
export const main = async ({ streamArn }) => {
  const client = new KinesisClient({});
  try {
    await client.send(
      new PutRecordsCommand({
        StreamARN: streamArn,
        Records: [
          {
            Data: new Uint8Array(),
            /**
             * Determines which shard in the stream the data record is assigned to.
             * Partition keys are Unicode strings with a maximum length limit of 256
             * characters for each key. Amazon Kinesis Data Streams uses the
            partition
             * key as input to a hash function that maps the partition key and
             * associated data to a specific shard.
             */
            PartitionKey: "TEST_KEY",
          },
          {
            Data: new Uint8Array(),
            PartitionKey: "TEST_KEY",
          },
        ],
      }),
    );
  } catch (caught) {
```

```

    if (caught instanceof Error) {
      //
    } else {
      throw caught;
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    streamArn: {
      type: "string",
      description: "The ARN of the stream.",
    },
  };

  const { values } = parseArgs({ options });
  main(values);
}

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [PutRecords](#)를 참조하세요.

서버리스 예제

Kinesis 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제에서는 Kinesis 스트림에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 Kinesis 페이로드를 검색하고, Base64에서 디코딩하고, 레코드 콘텐츠를 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 Kinesis 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

TypeScript를 사용하여 Lambda로 Kinesis 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});
```

```

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

Kinesis 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제는 Kinesis 스트림에서 이벤트를 수신하는 Lambda 함수에 대한 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Javascript를 사용하여 Lambda로 Kinesis 배치 항목 실패 보고

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.
      Lambda will immediately begin to retry processing from this failed item
      onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

TypeScript를 사용하여 Lambda로 Kinesis 배치 항목 실패 보고

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
```

```

} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.
      Lambda will immediately begin to retry processing from this failed item
      onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

SDK for JavaScript (v3)를 사용한 Lambda 예제

다음 코드 예제에서는 Lambda와 함께 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)
- [시나리오](#)
- [서버리스 예제](#)

시작하기

Hello Lambda

다음 코드 예제에서는 Lambda 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListFunctions](#)를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- IAM 역할과 Lambda 함수를 생성하고 핸들러 코드를 업로드합니다.
- 단일 파라미터로 함수를 간접적으로 간접 호출하고 결과를 가져옵니다.
- 함수 코드를 업데이트하고 환경 변수로 구성합니다.
- 새 파라미터로 함수를 간접적으로 간접 호출하고 결과를 가져옵니다. 반환된 실행 로그를 표시합니다.
- 계정의 함수를 나열합니다.

자세한 내용은 [콘솔로 Lambda 함수 생성](#)을 참조하세요.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Lambda에 로그에 쓸 수 있는 권한을 부여하는 AWS Identity and Access Management (IAM) 역할을 생성합니다.

```
logger.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 간접 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,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

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 항목을 참조하세요.

- [CreateFunction](#)
- [DeleteFunction](#)
- [GetFunction](#)
- [간접 호출](#)
- [ListFunctions](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

작업

CreateFunction

다음 코드 예시는 CreateFunction의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateFunction](#)을 참조하세요.

DeleteFunction

다음 코드 예시는 DeleteFunction의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteFunction](#)을 참조하세요.

GetFunction

다음 코드 예시는 GetFunction의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetFunction](#)을 참조하세요.

Invoke

다음 코드 예시는 Invoke의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 [간접 호출](#)을 참조하세요.

ListFunctions

다음 코드 예시는 ListFunctions의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListFunctions](#)를 참조하세요.

UpdateFunctionCode

다음 코드 예시는 UpdateFunctionCode의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [UpdateFunctionCode](#)를 참조하세요.

UpdateFunctionConfiguration

다음 코드 예시는 UpdateFunctionConfiguration의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateFunctionConfiguration](#)을 참조하세요.

시나리오

Lambda 함수를 사용하여 알려진 사용자를 자동으로 확인

다음 코드 예제는 Lambda 함수를 사용하여 알려진 Amazon Cognito 사용자를 자동으로 확인하는 방법을 보여줍니다.

- PreSignUp 트리거에 대해 Lambda 함수를 호출하도록 사용자 풀을 구성합니다.
- Amazon Cognito를 사용하여 사용자 가입시키기
- Lambda 함수는 DynamoDB 테이블을 스캔하고 알려진 사용자를 자동으로 확인합니다.
- 새 사용자로 로그인한 다음 리소스를 정리합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

대화형 '시나리오' 실행을 구성합니다. JavaScript(v3) 예제에서는 시나리오 실행기를 공유하여 복잡한 예제를 간소화합니다. 전체 소스 코드는 GitHub에 있습니다.

```
import { AutoConfirm } from "../scenario-auto-confirm.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
```

```

        UserName: "test_user_1",
        Email: "test_email_1@example.com",
    },
    {
        UserName: "test_user_2",
        Email: "test_email_2@example.com",
    },
    {
        UserName: "test_user_3",
        Email: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito authentication behavior.",
  });
}

```

이 시나리오에서는 알려진 사용자를 자동으로 확인하는 방법을 보여줍니다. 여기에서는 예제 단계를 오케스트레이션합니다.

```

import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {

```

```

    Scenario,
    ScenarioAction,
    ScenarioInput,
    ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
    getStackOutputs,
    logCleanupReminder,
    promptForStackName,
    promptForStackRegion,
    skipWhenErrors,
} from "../steps-common.js";
import { populateTable } from "../actions/dynamodb-actions.js";
import {
    addPreSignUpHandler,
    deleteUser,
    getUser,
    signIn,
    signUpUser,
} from "../actions/cognito-actions.js";
import {
    getLatestLogStreamForLambda,
    getLogEvents,
} from "../actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { Username: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

```

```
const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);
```

```

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (** @type {State} */ state) => {
    const [, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool \
"${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,

```

```
        userPoolId: state.UserPoolId,
        username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
        // Do nothing. We're not expecting the user to exist before
        // sign up is complete.
        return;
    }

    if (err) {
        state.errors.push(err);
        return;
    }

    if (user) {
        state.errors.push(
            new Error(
                `The user "${state.selectedUser}" already exists in the user pool "${state.UserPoolId}".`,
            ),
        );
    }
},
{
    skipWhen: skipWhenErrors,
},
);

const createPassword = new ScenarioInput(
    "password",
    "Enter a password that has at least eight characters, uppercase, lowercase, numbers and symbols.",
    { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
    "logSignUpExistingUser",
    (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
    { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
    "signUpExistingUser",
```

```

async (/** @type {State} */ state) => {
  const signUp = (password) =>
    signUpUser({
      region: state.stackRegion,
      userPoolClientId: state.UserPoolClientId,
      username: state.selectedUser,
      email: state.users.find((u) => u.UserName === state.selectedUser)
        .UserEmail,
      password,
    });

  let [_, err] = await signUp(state.password);

  while (err?.name === "InvalidPasswordException") {
    console.warn("The password you entered was invalid.");
    await createPassword.handle(state);
    [_, err] = await signUp(state.password);
  }

  if (err) {
    state.errors.push(err);
  }
},
{ skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"$${state.selectedUser} was signed up successfully.`",
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (/** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
  }
);

```



```
});
if (logStreamErr) {
  state.errors.push(logStreamErr);
  return;
}

console.log(
  `Getting some recent events from log stream "${logStream.logStreamName}"`,
);
const [logEvents, logEventsErr] = await getLogEvents({
  functionName: state.AutoConfirmHandlerName,
  region: state.stackRegion,
  eventCount: 10,
  logStreamName: logStream.logStreamName,
});
if (logEventsErr) {
  state.errors.push(logEventsErr);
  return;
}

console.log(logEvents.map((ev) => `t${ev.message}`).join(""));
},
{ skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });
  });

if (err?.name === "PasswordResetRequiredException") {
  state.errors.push(new Error("Please reset your password."));
  return;
}
```

```
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with: ${state.token.slice(0, 11)}`,
  { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [_, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);
```

```
const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State} */ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
```

```
);
```

다음은 다른 시나리오와 공유되는 단계입니다.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);
```

```
export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);
```

Lambda 함수에서 PreSignUp 트리거에 대한 핸들러.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "../user-repository";
import { DynamoDBUserRepository } from "../user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user '${event.userName}'`,
    );

    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }

    const eventEmail = this.getEventUserEmail(event);
    console.log(`Looking up email ${eventEmail}.`);
    const storedUserInfo =
```

```
    await this.userRepository.getUserInfoByEmail(eventEmail);

    if (!storedUserInfo) {
      console.log(
        `Email ${eventEmail} not found. Email verification is required.`
      );
      return event;
    }

    if (storedUserInfo.UserName !== event.userName) {
      console.log(
        `UserEmail ${eventEmail} found, but stored UserName
        '${storedUserInfo.UserName}' does not match supplied UserName '${event.userName}'.
        Verification is required.`
      );
    } else {
      console.log(
        `UserEmail ${eventEmail} found with matching UserName
        ${storedUserInfo.UserName}. User is confirmed.`
      );
      event.response.autoConfirmUser = true;
      event.response.autoVerifyEmail = true;
    }
    return event;
  }
}

const createPreSignUpHandler = (): PreSignUpHandler => {
  const tableName = process.env.TABLE_NAME;
  if (!tableName) {
    throw new Error("TABLE_NAME environment variable is not set");
  }

  const userRepository = new DynamoDBUserRepository(tableName);
  return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

CloudWatch Logs 작업의 모듈.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
  unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,

```

```

*   region: string
* }} config
* @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[] |
null, unknown]>}
*/
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};

```

Amazon Cognito 작업 모듈.

```

import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool

```



```

* @param {{ region: string, userPoolId: string, handlerArn: string }} config
* @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").UpdateUserPoolCommandOutput | null, unknown]>}
*/
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
* Attempt to register a user to a user pool with a given username and password.
* @param {{
*   region: string,
*   userPoolClientId: string,
*   username: string,
*   email: string,
*   password: string
* }} config
* @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").SignUpCommandOutput | null, unknown]>}
*/
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,

```

```

    email,
    password,
  }) => {
    try {
      const cognitoClient = new CognitoIdentityProviderClient({
        region,
      });

      const response = await cognitoClient.send(
        new SignUpCommand({
          ClientId: userPoolClientId,
          Username: username,
          Password: password,
          UserAttributes: [{ Name: "email", Value: email }],
        }),
      );
      return [response, null];
    } catch (err) {
      return [null, err];
    }
  };

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 * flow.
 * @param {{ region: string, clientId: string, username: string, password: string }}
 * config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
 * provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
}

```

```

};

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

```

DynamoDB 작업의 모듈.

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

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }}
  config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
  null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.

- [DeleteUser](#)
- [InitiateAuth](#)
- [SignUp](#)

- [UpdateUserPool](#)

사진을 관리하기 위한 서버리스 애플리케이션 만들기

다음 코드 예시에서는 사용자가 레이블을 사용하여 사진을 관리할 수 있는 서버리스 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 [AWS 커뮤니티](#)의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

고객 피드백 분석을 위한 애플리케이션 생성

다음 코드 예제에서는 고객 의견 카드를 분석하고, 원어에서 번역하고, 감정을 파악하고, 번역된 텍스트에서 오디오 파일을 생성하는 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 코드와 배포 지침은 [GitHub](#)의 프로젝트를 참조하세요. 다음 발췌문은 Lambda 함수 내에서 AWS SDK for JavaScript가 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 예제에서 사용되는 서비스

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

브라우저에서 Lambda 함수 간접 호출

다음 코드 예제에서는 브라우저에서 AWS Lambda 함수를 호출하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS Lambda 함수를 사용하여 사용자 선택 항목으로 Amazon DynamoDB 테이블을 업데이트하는 브라우저 기반 애플리케이션을 생성할 수 있습니다. 이 앱은 AWS SDK for JavaScript v3를 사용합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB

- Lambda

API Gateway를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon API Gateway에서 호출한 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

Lambda JavaScript 런타임 API를 사용하여 AWS Lambda 함수를 생성하는 방법을 보여줍니다. 이 예제에서는 다양한 AWS 서비스를 호출하여 특정 사용 사례를 수행합니다. 이 예제에서는 Amazon API Gateway에서 간접 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

예약된 이벤트를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon EventBridge 예약 이벤트에서 호출된 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS Lambda 함수를 호출하는 Amazon EventBridge 예약 이벤트를 생성하는 방법을 보여줍니다. Lambda 함수가 간접 호출될 때 cron 표현식을 사용하여 일정을 예약하도록 EventBridge를 구성합니다. 이 예제에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

서버리스 예제

Lambda 함수를 사용하여 Amazon RDS 데이터베이스에 연결

다음 코드 예제는 RDS 데이터베이스에 연결하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 간단한 데이터베이스 요청을 하고 결과를 반환합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda 함수에서 Amazon RDS 데이터베이스에 연결

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {
```

```
    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,
  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;
}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

TypeScript를 사용하여 Lambda 함수에서 Amazon RDS 데이터베이스에 연결

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
// DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!

async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
    } catch (err) {
        console.error(err);
    }
}
```

```

    });
    const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
    console.log('result:', rows);
    return rows;
  }
  catch (err) {
    console.log(err);
  }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
string }> => {
  // Execute database flow
  const result = await dbOps();

  // Return error is result is undefined
  if (result == undefined)
    return {
      statusCode: 500,
      body: JSON.stringify(`Error with connection to DB host`)
    }

  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
  };
};

```

Kinesis 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제에서는 Kinesis 스트림에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 Kinesis 페이로드를 검색하고, Base64에서 디코딩하고, 레코드 콘텐츠를 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 Kinesis 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

TypeScript를 사용하여 Lambda로 Kinesis 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});
```

```

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

DynamoDB 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제에서는 DynamoDB 스트림에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DynamoDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

TypeScript를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Amazon DocumentDB 트리거에서 간접적으로 Lambda 함수 호출

다음 코드 예제에서는 DocumentDB 변경 스트림에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DocumentDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 Amazon DocumentDB 이벤트 소비

```
console.log('Loading function');
exports.handler = async (event, context) => {
  event.events.forEach(record => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record) => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
    2));
};
```

TypeScript를 사용하여 Lambda로 Amazon DocumentDB 이벤트 소비

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
};
```

```

    return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};

```

Amazon MSK 트리거를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon MSK 클러스터에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 MSK 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 Amazon MSK 이벤트를 사용합니다.

```

exports.handler = async (event) => {
  // Iterate through keys
  for (let key in event.records) {
    console.log('Key: ', key)
    // Iterate through records
    event.records[key].map((record) => {
      console.log('Record: ', record)
      // Decode base64
      const msg = Buffer.from(record.value, 'base64').toString()
      console.log('Message:', msg)
    })
  }
}

```

TypeScript를 사용하여 Lambda로 Amazon MSK 이벤트를 사용합니다.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    }
  }
}
```

Amazon S3 트리거를 사용하여 Lambda 함수 간접 호출

다음 코드 예제는 S3 버킷에 객체를 업로드하여 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 해당 함수는 이벤트 파라미터에서 S3 버킷 이름과 객체 키를 검색하고 Amazon S3 API를 호출하여 객체의 콘텐츠 유형을 검색하고 로깅합니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 S3 이벤트를 사용합니다.

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

const client = new S3Client();

export const handler = async (event, context) => {

  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));

  try {
    const { ContentType } = await client.send(new HeadObjectCommand({
      Bucket: bucket,
      Key: key,
    }));

    console.log('CONTENT TYPE:', ContentType);
    return ContentType;

  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

TypeScript를 사용하여 Lambda로 S3 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

Amazon SNS 트리거를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 SNS 주제의 메시지를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 함수는 이벤트 파라미터에서 메시지를 검색하고 각 메시지의 내용을 로깅합니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 SNS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

TypeScript를 사용하여 Lambda로 SNS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
```

```

    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}

```

Amazon SQS 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제는 SQS 대기열에서 메시지를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 함수는 이벤트 파라미터에서 메시지를 검색하고 각 메시지의 내용을 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 SQS 이벤트 사용

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

```



```

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}

```

TypeScript를 사용하여 Lambda로 SQS 이벤트 사용

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}

```

Kinesis 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제는 Kinesis 스트림에서 이벤트를 수신하는 Lambda 함수에 대한 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Javascript를 사용하여 Lambda로 Kinesis 배치 항목 실패 보고

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.
      Lambda will immediately begin to retry processing from this failed item
      onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
```

```

    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}

```

TypeScript를 사용하여 Lambda로 Kinesis 배치 항목 실패 보고

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    KinesisStreamEvent,
    Context,
    KinesisStreamHandler,
    KinesisStreamRecordPayload,
    KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
    logLevel: "INFO",
    serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
    event: KinesisStreamEvent,
    context: Context
): Promise<KinesisStreamBatchResponse> => {
    for (const record of event.Records) {
        try {
            logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
            const recordData = await getRecordDataAsync(record.kinesis);
            logger.info(`Record Data: ${recordData}`);
            // TODO: Do interesting work based on the new data
        } catch (err) {
            logger.error(`An error occurred ${err}`);
            /* Since we are working with streams, we can return the failed item
            immediately.
            Lambda will immediately begin to retry processing from this failed item
            onwards. */
            return {
                batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
            };
        }
    }
}

```

```

    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

DynamoDB 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제에서는 DynamoDB 스트림에서 이벤트를 수신하는 Lambda 함수에 대한 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```

export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
    }
  }
}

```

```

        return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
    }
}

return { batchItemFailures: [] };
};

```

TypeScript를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```

import {
    DynamoDBBatchResponse,
    DynamoDBBatchItemFailure,
    DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
    event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
    const batchItemFailures: DynamoDBBatchItemFailure[] = [];
    let curRecordSequenceNumber;

    for (const record of event.Records) {
        curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

        if (curRecordSequenceNumber) {
            batchItemFailures.push({
                itemIdentifier: curRecordSequenceNumber,
            });
        }
    }

    return { batchItemFailures: batchItemFailures };
};

```

Amazon SQS 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제는 SQS 대기열에서 이벤트를 수신하는 Lambda 함수에 대한 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda에서 SQS 배치 항목 실패를 보고합니다.

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
  const batchItemFailures = [];
  for (const record of event.Records) {
    try {
      await processMessageAsync(record, context);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }
  return { batchItemFailures };
};

async function processMessageAsync(record, context) {
  if (record.body && record.body.includes("error")) {
    throw new Error("There is an error in the SQS Message.");
  }
  console.log(`Processed message: ${record.body}`);
}
```

TypeScript를 사용하여 Lambda로 SQS 배치 항목 실패를 보고합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
  'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
  Promise<SQSBatchResponse> => {
  const batchItemFailures: SQSBatchItemFailure[] = [];

  for (const record of event.Records) {
```

```
    try {
      await processMessageAsync(record);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }

  return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
  if (record.body && record.body.includes("error")) {
    throw new Error('There is an error in the SQS Message.');
```

SDK for JavaScript(v3)를 사용한 Amazon Lex 예제

다음 코드 예제에서는 Amazon Lex에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

Amazon Lex 챗봇 구축

다음 코드 예제에서는 챗봇을 만들어 웹사이트 방문자를 참여시키는 방법을 보여줍니다.

SDK for JavaScript(v3)

Amazon Lex API를 사용하여 웹 애플리케이션 내에 챗봇을 구축하여 웹 사이트 방문자의 참여를 유도하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 AWS SDK for JavaScript 개발자 안내서의 [Amazon Lex 챗봇 구축](#) 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

SDK for JavaScript(v3)를 사용한 Amazon Location 예제

다음 코드 예제에서는 Amazon Location과 함께 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)

시작하기

Amazon Location 시작

다음 코드 예제에서는 Amazon Location Service 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node:url";
import {
  LocationClient,
  ListGeofenceCollectionsCommand,
} from "@aws-sdk/client-location";

/**
 * Lists geofences from a specified geofence collection asynchronously.
 */
export const main = async () => {
  const region = "eu-west-1";
  const locationClient = new LocationClient({ region: region });
  const listGeofenceCollParams = {
    MaxResults: 100,
  };
  try {
    const command = new ListGeofenceCollectionsCommand(listGeofenceCollParams);
    const response = await locationClient.send(command);
    const geofenceEntries = response.Entries;
    if (geofenceEntries.length === 0) {
      console.log("No Geofences were found in the collection.");
    } else {
      for (const geofenceEntry of geofenceEntries) {
        console.log(`Geofence ID: ${geofenceEntry.CollectionName}`);
      }
    }
  } catch (error) {
    console.error(
      `A validation error occurred while creating geofence: ${error} \n Exiting program.`
    );
    return;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [ListGeofenceCollections](#)
 - [ListGeofences](#)

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon Location 맵을 만듭니다.
- Amazon Location API 키를 만듭니다.
- 맵 URL을 표시합니다.
- 지오펜스 컬렉션을 만듭니다.
- 지오펜스 지오메트리를 저장합니다.
- 트래커 리소스를 만듭니다.
- 디바이스의 위치를 업데이트합니다.
- 지정된 디바이스에 대한 최신 위치 업데이트를 검색합니다.
- 경로 계산기를 만듭니다.
- 시애틀과 밴쿠버 사이의 거리를 결정합니다.
- Amazon Location 상위 수준 API를 사용합니다.
- Amazon Location 자산을 삭제합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/*
Before running this JavaScript code example, set up your development environment,
including your credentials.
This demo illustrates how to use the AWS SDK for JavaScript (v3) to work with Amazon
Location Service.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/getting-started.html
*/

import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

import {
  CreateMapCommand,
  CreateGeofenceCollectionCommand,
  PutGeofenceCommand,
  CreateTrackerCommand,
  BatchUpdateDevicePositionCommand,
  GetDevicePositionCommand,
  CreateRouteCalculatorCommand,
  CalculateRouteCommand,
  LocationClient,
  ConflictException,
  ResourceNotFoundException,
  DeleteGeofenceCollectionCommand,
  DeleteRouteCalculatorCommand,
  DeleteTrackerCommand,
  DeleteMapCommand,
} from "@aws-sdk/client-location";

import {
  GeoPlacesClient,
  ReverseGeocodeCommand,
  SearchNearbyCommand,
  SearchTextCommand,
  GetPlaceCommand,
  ValidationException,
```

```

} from "@aws-sdk/client-geo-places";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";

/*The inputs for this example can be edited in the ./input.json.*/
import data from "./inputs.json" with { type: "json" };

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
/* v8 ignore next 3 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
  verbose: "false",
});

const pressEnterConfirm = new ScenarioInput(
  "confirm",
  "Press Enter to continue",
  {
    type: "confirm",
    verbose: "false",
  },
);

const region = "eu-west-1";

const locationClient = new LocationClient({ region: region });

const greet = new ScenarioOutput(
  "greet",
  "Welcome to the Amazon Location Use demo! \n" +
    "AWS Location Service is a fully managed service offered by Amazon Web Services (AWS) that " +
    "provides location-based services for developers. This service simplifies " +
    "the integration of location-based features into applications, making it " +
    "Maps: The service provides access to high-quality maps, satellite imagery, " +
    "and geospatial data from various providers, allowing developers to " +
    "easily embed maps into their applications:\n" +
    "Tracking: The Location Service enables real-time tracking of mobile devices, "
  +
  "assets, or other entities, allowing developers to build applications " +

```

```

    "that can monitor the location of people, vehicles, or other objects.\n" +
    "Geocoding: The service provides the ability to convert addresses or " +
    "location names into geographic coordinates (latitude and longitude), " +
    "and vice versa, enabling developers to integrate location-based search " +
    "and routing functionality into their applications. " +
    "Please define values ./inputs.json for each user-defined variable used in this
app. Otherwise the default is used:\n" +
    "- mapName: The name of the map to be create (default is 'AWSMap').\n" +
    "- keyName: The name of the API key to create (default is ' AWSApiKey')\n" +
    "- collectionName: The name of the geofence collection (default is
'AWSLocationCollection')\n" +
    "- geoId: The geographic identifier used for the geofence or map (default is
'geoId')\n" +
    "- trackerName: The name of the tracker (default is 'geoTracker')\n" +
    "- calculatorName: The name of the route calculator (default is
'AWSRouteCalc')\n" +
    "- deviceId: The ID of the device (default is 'iPhone-112356')",

    { header: true },
);
const displayCreateAMap = new ScenarioOutput(
    "displayCreateAMap",
    "1. Create a map\n" +
    "An AWS Location map can enhance the user experience of your " +
    " application by providing accurate and personalized location-based " +
    " features. For example, you could use the geocoding capabilities to " +
    " allow users to search for and locate businesses, landmarks, or " +
    " other points of interest within a specific region.",
);

const sdkCreateAMap = new ScenarioAction(
    "sdkCreateAMap",
    async (** @type {State} */ state) => {
        const createMapParams = {
            MapName: `${data.inputs.mapName}`,
            Configuration: { style: "VectorEsriNavigation" },
        };
        try {
            const command = new CreateMapCommand(createMapParams);
            const response = await locationClient.send(command);
            state.MapName = response.MapName;
            console.log("Map created. Map ARN is: ", state.MapName);
        } catch (error) {
            console.error("Error creating map: ", error);
        }
    }
);

```

```

        throw error;
    }
},
);

const displayMapUrl = new ScenarioOutput(
    "displayMapUrl",
    "2. Display Map URL\n" +
        "When you embed a map in a web app or website, the API key is " +
        "included in the map tile URL to authenticate requests. You can " +
        "restrict API keys to specific AWS Location operations (e.g., only " +
        "maps, not geocoding). API keys can expire, ensuring temporary " +
        "access control.\n" +
        "In order to get the MAP URL you need to create and get the API Key value. " +
        "You can create and get the key value using the AWS Management Console under " +
        "Location Services. These operations cannot be completed using the " +
        "AWS SDK. For more information about getting the key value, see " +
        "the AWS Location Documentation.",
);

const sdkDisplayMapUrl = new ScenarioAction(
    "sdkDisplayMapUrl",
    async (** @type {State} */ state) => {
        const mapURL = `https://maps.geo.aws.amazon.com/maps/v0/maps/${state.MapName}/
tiles/{z}/{x}/{y}?key=API_KEY_VALUE`;
        state.mapURL = mapURL;
        console.log(
            `Replace \'API_KEY_VALUE\' in the following URL with the value for the API key
you create and get from the AWS Management Console under Location Services. This is
then the Map URL you can embed this URL in your Web app:\n
${state.mapURL}`,
        );
    },
);

const displayCreateGeoFenceColl = new ScenarioOutput(
    "displayCreateGeoFenceColl",
    "3. Create a geofence collection, which manages and stores geofences.",
);

const sdkCreateGeoFenceColl = new ScenarioAction(
    "sdkCreateGeoFenceColl",
    async (** @type {State} */ state) => {
        // Creates a new geofence collection.
        const geoFenceCollParams = {

```

```

        CollectionName: `${data.inputs.collectionName}`,
    };
    try {
        const command = new CreateGeofenceCollectionCommand(geoFenceCollParams);
        const response = await locationClient.send(command);
        state.CollectionName = response.CollectionName;
        console.log(
            `The geofence collection was successfully created: ${state.CollectionName}`,
        );
    } catch (caught) {
        if (caught instanceof ConflictException) {
            console.error(
                `An unexpected error occurred while creating the geofence collection:
                ${caught.message} \n Exiting program.`,
            );
            return;
        }
    }
},
);
const displayStoreGeometry = new ScenarioOutput(
    "displayStoreGeometry",
    "4. Store a geofence geometry in a given geofence collection. " +
    "An AWS Location geofence is a virtual boundary that defines a geographic area "
    +
    "on a map. It is a useful feature for tracking the location of " +
    "assets or monitoring the movement of objects within a specific region. " +
    "To define a geofence, you need to specify the coordinates of a " +
    "polygon that represents the area of interest. The polygon must be " +
    "defined in a counter-clockwise direction, meaning that the points of " +
    "the polygon must be listed in a counter-clockwise order. " +
    "This is a requirement for the AWS Location service to correctly " +
    "interpret the geofence and ensure that the location data is " +
    "accurately processed within the defined area.",
);

const sdkStoreGeometry = new ScenarioAction(
    "sdkStoreGeometry",
    async (/** @type {State} */ state) => {
        const geoFenceGeoParams = {
            CollectionName: `${data.inputs.collectionName}`,
            GeofenceId: `${data.inputs.geoId}`,
            Geometry: {
                Polygon: [

```

```

        [
            [-122.3381, 47.6101],
            [-122.3281, 47.6101],
            [-122.3281, 47.6201],
            [-122.3381, 47.6201],
            [-122.3381, 47.6101],
        ],
    ],
},
};
try {
    const command = new PutGeofenceCommand(geoFenceGeoParams);
    const response = await locationClient.send(command);
    state.GeoFencId = response.GeofenceId;
    console.log("GeoFence created. GeoFence ID is: ", state.GeoFencId);
} catch (caught) {
    if (caught instanceof ValidationException) {
        console.error(
            `A validation error occurred while creating geofence: ${caught.message} \n
Exiting program.`
        );
        return;
    }
}
},
);
const displayCreateTracker = new ScenarioOutput(
    "displayCreateTracker",
    "5. Create a tracker resource which lets you retrieve current and historical location of devices.",
);

const sdkCreateTracker = new ScenarioAction(
    "sdkCreateTracker",
    async (** @type {State} */ state) => {
        //Creates a new tracker resource in your AWS account, which you can use to track the location of devices.
        const createTrackerParams = {
            TrackerName: `${data.inputs.trackerName}`,
            Description: "Created using the JavaScript V3 SDK",
            PositionFiltering: "TimeBased",
        };
        try {
            const command = new CreateTrackerCommand(createTrackerParams);

```



```

    const response = await locationClient.send(command);
    state.trackerName = response.TrackerName;
    console.log("Tracker created. Tracker name is : ", state.trackerName);
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `A validation error occurred while creating geofence: ${caught.message} \n
Exiting program.`
      );
    } else {
      console.error(
        `An unexpected error error occurred: ${caught.message} \n Exiting program.`
      );
    }
    return;
  }
},
);
const displayUpdatePosition = new ScenarioOutput(
  "displayUpdatePosition",
  "6. Update the position of a device in the location tracking system." +
  "The AWS Location Service does not enforce a strict format for deviceId, but it must:\n " +
  "- Be a string (case-sensitive).\n" +
  "- Be 1-100 characters long.\n" +
  "- Contain only: Alphanumeric characters (A-Z, a-z, 0-9); Underscores (_); Hyphens (-); and be the same ID used when sending and retrieving positions.",
);

const sdkUpdatePosition = new ScenarioAction(
  "sdkUpdatePosition",
  async (/** @type {State} */ state) => {
    // Updates the position of a device in the location tracking system.

    const updateDevicePosParams = {
      TrackerName: `${data.inputs.trackerName}`,
      Updates: [
        {
          DeviceId: `${data.inputs.deviceId}`,
          SampleTime: new Date(),
          Position: [-122.4194, 37.7749],
        },
      ],
    };

    try {
      const command = new BatchUpdateDevicePositionCommand(

```

```

        updateDevicePosParams,
    );
    const response = await locationClient.send(command);
    console.log(
        `Device with id ${data.inputs.deviceId} was successfully updated in the
location tracking system. `,
    );
} catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
        console.error(
            `A validation error occurred while updating the device: ${caught.message}
\n Exiting program.`,
        );
    }
}
},
);
const displayRetrievePosition = new ScenarioOutput(
    "displayRetrievePosition",
    "7. Retrieve the most recent position update for a specified device.",
);

const sdkRetrievePosition = new ScenarioAction(
    "sdkRetrievePosition",
    async (/** @type {State} */ state) => {
        const devicePositionParams = {
            TrackerName: `${data.inputs.trackerName}`,
            DeviceId: `${data.inputs.deviceId}`,
        };
        try {
            const command = new GetDevicePositionCommand(devicePositionParams);
            const response = await locationClient.send(command);
            state.position = response.Position;
            console.log("Successfully fetched device position: : ", state.position);
        } catch (caught) {
            if (caught instanceof ResourceNotFoundException) {
                console.error(
                    `"The resource was not found: ${caught.message} \n Exiting program.`,
                );
            } else {
                `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
            }
            return;
        }
    }
);

```

```
    },
  );
  const displayCreateRouteCalc = new ScenarioOutput(
    "displayCreateRouteCalc",
    "8. Create a route calculator.",
  );

  const sdkCreateRouteCalc = new ScenarioAction(
    "sdkCreateRouteCalc",
    async (** @type {State} */ state) => {
      const routeCalcParams = {
        CalculatorName: `${data.inputs.calculatorName}`,
        DataSource: "Esri",
      };
      try {
        // Creates a new route calculator with the specified name and data source.
        const command = new CreateRouteCalculatorCommand(routeCalcParams);
        const response = await locationClient.send(command);
        state.CalculatorName = response.CalculatorName;
        console.log(
          "Route calculator created successfully. Calculator name is: ",
          state.CalculatorName,
        );
      } catch (caught) {
        if (caught instanceof ConflictException) {
          console.error(
            `An conflict occurred: ${caught.message} \n Exiting program.`
          );
          return;
        }
      }
    },
  );

  const displayDetermineDist = new ScenarioOutput(
    "displayDetermineDist",
    "9. Determine the distance between Seattle and Vancouver using the route calculator.",
  );

  const sdkDetermineDist = new ScenarioAction(
    "sdkDetermineDist",
    async (** @type {State} */ state) => {
      // Calculates the distance between two locations asynchronously.
      const determineDist = {
```

```

    CalculatorName: `${data.inputs.calculatorName}`,
    DeparturePosition: [-122.3321, 47.6062],
    DestinationPosition: [-123.1216, 49.2827],
    TravelMode: "Car",
    DistanceUnit: "Kilometers",
  };
  try {
    const command = new CalculateRouteCommand(determineDist);
    const response = await locationClient.send(command);

    console.log(
      "Successfully calculated route. The distance in kilometers is : ",
      response.Summary.Distance,
    );
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `Failed to calculate route: ${caught.message} \n Exiting program.`
      );
    }
    return;
  }
},
);
const displayUseGeoPlacesClient = new ScenarioOutput(
  "displayUseGeoPlacesClient",
  "10. Use the GeoPlacesAsyncClient to perform additional operations. " +
  "This scenario will show use of the GeoPlacesClient that enables" +
  "location search and geocoding capabilities for your applications. " +
  "We are going to use this client to perform these AWS Location tasks: \n" +
  " - Reverse Geocoding (reverseGeocode): Converts geographic coordinates into\n\n" +
  "addresses.\n\n" +
  " - Place Search (searchText): Finds places based on search queries.\n\n" +
  " - Nearby Search (searchNearby): Finds places near a specific location.\n\n" +
  "First we will perform a Reverse Geocoding operation",
);
const sdkUseGeoPlacesClient = new ScenarioAction(
  "sdkUseGeoPlacesClient",
  async (/** @type {State} */ state) => {
    const geoPlacesClient = new GeoPlacesClient({ region: region });

    const reverseGeoCodeParams = {
      QueryPosition: [-122.4194, 37.7749],

```

```
};
const searchTextParams = {
  QueryText: "coffee shop",
  BiasPosition: [-122.4194, 37.7749], //San Fransisco
};
const searchNearbyParams = {
  QueryPosition: [-122.4194, 37.7749],
  QueryRadius: Number("1000"),
};
try {
  /* Performs reverse geocoding using the AWS Geo Places API.
  Reverse geocoding is the process of converting geographic coordinates (latitude
  and longitude) to a human-readable address.
  This method uses the latitude and longitude of San Francisco as the input, and
  prints the resulting address.*/

  console.log("Use latitude 37.7749 and longitude -122.4194.");
  const command = new ReverseGeocodeCommand(reverseGeoCodeParams);
  const response = await geoPlacesClient.send(command);
  console.log(
    "Successfully calculated route. The distance in kilometers is : ",
    response.ResultItems[0].Distance,
  );
} catch (caught) {
  if (caught instanceof ValidationException) {
    console.error(
      `An conflict occurred: ${caught.message} \n Exiting program.`
    );
    return;
  }
}
try {
  console.log(
    "Now we are going to perform a text search using coffee shop",
  );

  /*Searches for a place using the provided search query and prints the detailed
  information of the first result.
  @param searchTextParams the search query to be used for the place search (ex,
  coffee shop)*/

  const command = new SearchTextCommand(searchTextParams);
  const response = await geoPlacesClient.send(command);
  const placeId = response.ResultItems[0].PlaceId.toString();
```

```
const getPlaceCommand = new GetPlaceCommand({
  PlaceId: placeId,
});
const getPlaceResponse = await geoPlacesClient.send(getPlaceCommand);
console.log(
  `Detailed Place Information: \n Name and address:
  ${getPlaceResponse.Address.Label}`,
);

const foodTypes = getPlaceResponse.FoodTypes;
if (foodTypes.length) {
  console.log("Food Types: ");
  for (const foodType of foodTypes) {
    console.log("- ", foodType.LocalizedName);
  }
} else {
  console.log("No food types available.");
}
} catch (caught) {
  if (caught instanceof ValidationException) {
    console.error(
      `An conflict occurred: ${caught.message} \n Exiting program.`,
    );
    return;
  }
}
try {
  console.log("\nNow we are going to perform a nearby search.");
  const command = new SearchNearbyCommand(searchNearbyParams);
  const response = await geoPlacesClient.send(command);
  const resultItems = response.ResultItems;
  console.log("\nSuccessfully performed nearby search.");
  for (const resultItem of resultItems) {
    console.log("Name and address: ", resultItem.Address.Label);
    console.log("Distance: ", resultItem.Distance);
  }
} catch (caught) {
  if (caught instanceof ValidationException) {
    console.error(
      `An conflict occurred: ${caught.message} \n Exiting program.`,
    );
    return;
  }
}
```

```
    },
  );

const displayDeleteResources = new ScenarioOutput(
  "displayDeleteResources",
  "11. Delete the AWS Location Services resources. " +
    "Would you like to delete the AWS Location Services resources? (y/n)",
);

const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    const deleteGeofenceCollParams = {
      CollectionName: `${state.CollectionName}`,
    };
    const deleteRouteCalculatorParams = {
      CalculatorName: `${state.CalculatorName}`,
    };
    const deleteTrackerParams = { TrackerName: `${state.trackerName}` };
    const deleteMapParams = { MapName: `${state.MapName}` };
    try {
      const command = new DeleteMapCommand(deleteMapParams);
      const response = await locationClient.send(command);
      console.log("Map deleted.");
    } catch (error) {
      console.log("Error deleting map: ", error);
    }
    try {
      const command = new DeleteGeofenceCollectionCommand(
        deleteGeofenceCollParams,
      );
      const response = await locationClient.send(command);
      console.log("Geofence collection deleted.");
    } catch (error) {
      console.log("Error deleting geofence collection: ", error);
    }
    try {
      const command = new DeleteRouteCalculatorCommand(
        deleteRouteCalculatorParams,
      );
      const response = await locationClient.send(command);
      console.log("Route calculator deleted.");
    } catch (error) {
      console.log("Error deleting route calculator: ", error);
    }
  }
);
```

```
    }
    try {
      const command = new DeleteTrackerCommand(deleteTrackerParams);
      const response = await locationClient.send(command);
      console.log("Tracker deleted.");
    } catch (error) {
      console.log("Error deleting tracker: ", error);
    }
  },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "Thank you for checking out the Amazon Location Service Use demo. We hope you " +
    "learned something new, or got some inspiration for your own apps today!" +
    " For more Amazon Location Services examples in different programming languages, have a look at: " +
    "https://docs.aws.amazon.com/code-library/latest/ug/location_code_examples.html",
);

const myScenario = new Scenario("Location Services Scenario", [
  greet,
  pressEnter,
  displayCreateAMap,
  sdkCreateAMap,
  pressEnter,
  displayMapUrl,
  sdkDisplayMapUrl,
  pressEnter,
  displayCreateGeoFenceColl,
  sdkCreateGeoFenceColl,
  pressEnter,
  displayStoreGeometry,
  sdkStoreGeometry,
  pressEnter,
  displayCreateTracker,
  sdkCreateTracker,
  pressEnter,
  displayUpdatePosition,
  sdkUpdatePosition,
  pressEnter,
  displayRetrievePosition,
  sdkRetrievePosition,
```



```

    pressEnter,
    displayCreateRouteCalc,
    sdkCreateRouteCalc,
    pressEnter,
    displayDetermineDist,
    sdkDetermineDist,
    pressEnter,
    displayUseGeoPlacesClient,
    sdkUseGeoPlacesClient,
    pressEnter,
    displayDeleteResources,
    pressEnterConfirm,
    sdkDeleteResources,
    goodbye,
  ]);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}

```

작업

BatchUpdateDevicePosition

다음 코드 예시는 BatchUpdateDevicePosition의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node:url";
import {
  BatchUpdateDevicePositionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };
const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });
const updateDevicePosParams = {
  TrackerName: `${data.inputs.trackerName}`,
  Updates: [
    {
      DeviceId: `${data.inputs.deviceId}`,
      SampleTime: new Date(),
      Position: [-122.4194, 37.7749],
    },
  ],
};
export const main = async () => {
  try {
    const command = new BatchUpdateDevicePositionCommand(updateDevicePosParams);
    const response = await locationClient.send(command);
    //console.log("response ", response.Errors[0].Error);

    console.log(
      `Device with id ${data.inputs.deviceId} was successfully updated in the location tracking system. `,
      response,
    );
  } catch (error) {
    console.log("error ", error);
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [BatchUpdateDevicePosition](#)을 참조하세요.

CalculateRoute

다음 코드 예시는 CalculateRoute의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node:url";
import {
  CalculateRouteCommand,
  ResourceNotFoundException,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });

export const main = async () => {
  const routeCalcParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
    DeparturePosition: [-122.3321, 47.6062],
    DestinationPosition: [-123.1216, 49.2827],
    TravelMode: "Car",
    DistanceUnit: "Kilometers",
  };
  try {
    const command = new CalculateRouteCommand(routeCalcParams);
    const response = await locationClient.send(command);
```

```

    console.log(
      "Successfully calculated route. The distance in kilometers is : ",
      response.Summary.Distance,
    );
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `An conflict occurred: ${caught.message} \n Exiting program.` ,
      );
      return;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CalculateRoute](#)를 참조하세요.

CreateGeofenceCollection

다음 코드 예시는 CreateGeofenceCollection의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { fileURLToPath } from "node:url";
import {
  ConflictException,
  CreateGeofenceCollectionCommand,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const geoFenceCollParams = {

```

```

    CollectionName: `${data.inputs.collectionName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateGeofenceCollectionCommand(geoFenceCollParams);
    const response = await locationClient.send(command);
    console.log(
      "Collection created. Collection name is: ",
      response.CollectionName,
    );
  } catch (caught) {
    if (caught instanceof ConflictException) {
      console.error("A conflict occurred. Exiting program.");
      return;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateGeofenceCollection](#)을 참조하세요.

CreateMap

다음 코드 예시는 CreateMap의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { fileURLToPath } from "node:url";
import { CreateMapCommand, LocationClient } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const CreateMapCommandInput = {

```

```

    MapName: `${data.inputs.mapName}`,
    Configuration: { style: "VectorEsriNavigation" },
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateMapCommand(CreateMapCommandInput);
    const response = await locationClient.send(command);
    console.log("Map created. Map ARN is : ", response.MapArn);
  } catch (error) {
    console.error("Error creating map: ", error);
    throw error;
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateMap](#)을 참조하세요.

CreateRouteCalculator

다음 코드 예시는 CreateRouteCalculator의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { fileURLToPath } from "node:url";
import {
  ConflictException,
  CreateRouteCalculatorCommand,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });

export const main = async () => {

```

```

const routeCalcParams = {
  CalculatorName: `${data.inputs.calculatorName}`,
  DataSource: "Esri",
};
try {
  const command = new CreateRouteCalculatorCommand(routeCalcParams);
  const response = await locationClient.send(command);

  console.log(
    "Route calculator created successfully. Calculator name is ",
    response.CalculatorName,
  );
} catch (caught) {
  if (caught instanceof ConflictException) {
    console.error(
      `An conflict occurred: ${caught.message} \n Exiting program.`
    );
    return;
  }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateRouteCalculator](#)를 참조하세요.

CreateTracker

다음 코드 예시는 CreateTracker의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { fileURLToPath } from "node:url";
import { CreateTrackerCommand, LocationClient } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

```

```
const region = "eu-west-1";

export const main = async () => {
  const createTrackerParams = {
    TrackerName: `${data.inputs.trackerName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateTrackerCommand(createTrackerParams);
    const response = await locationClient.send(command);
    //state.trackerName - response.TrackerName;
    console.log("Tracker created. Tracker name is : ", response.TrackerName);
  } catch (error) {
    console.error("Error creating map: ", error);
    throw error;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateTracker](#)를 참조하세요.

DeleteGeofenceCollection

다음 코드 예시는 DeleteGeofenceCollection의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node:url";
import {
  DeleteGeofenceCollectionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "../inputs.json" with { type: "json" };
```



```
const region = "eu-west-1";

export const main = async () => {
  const deleteGeofenceCollParams = {
    CollectionName: `${data.inputs.collectionName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new DeleteGeofenceCollectionCommand(
      deleteGeofenceCollParams,
    );
    const response = await locationClient.send(command);
    console.log("Collection deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `${data.inputs.collectionName} Geofence collection not found.`
      );
      return;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteGeofenceCollection](#)을 참조하세요.

DeleteMap

다음 코드 예시는 DeleteMap의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node:url";
import {
  DeleteMapCommand,
```

```

    LocationClient,
    ResourceNotFoundException,
  } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteMapParams = {
    MapName: `${data.inputs.mapName}`,
  };
  try {
    const locationClient = new LocationClient({ region: region });
    const command = new DeleteMapCommand(deleteMapParams);
    const response = await locationClient.send(command);
    console.log("Map deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(`${data.inputs.mapName} map not found.`);
      return;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteMap](#)을 참조하세요.

DeleteRouteCalculator

다음 코드 예시는 DeleteRouteCalculator의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node:url";
```

```
import {
  DeleteRouteCalculatorCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteRouteCalculatorParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
  };
  try {
    const locationClient = new LocationClient({ region: region });
    const command = new DeleteRouteCalculatorCommand(
      deleteRouteCalculatorParams,
    );
    const response = await locationClient.send(command);
    console.log("Route calculator deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `${data.inputs.calculatorName} route calculator not found.`,
      );
      return;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteRouteCalculator](#)를 참조하세요.

DeleteTracker

다음 코드 예시는 DeleteTracker의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node:url";
import {
  DeleteTrackerCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteTrackerParams = {
    TrackerName: `${data.inputs.trackerName}`,
  };
  try {
    const locationClient = new LocationClient({ region: region });
    const command = new DeleteTrackerCommand(deleteTrackerParams);
    const response = await locationClient.send(command);
    console.log("Tracker deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(`${data.inputs.trackerName} tracker not found.`);
      return;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteTracker](#)를 참조하세요.

GetDevicePosition

다음 코드 예시는 GetDevicePosition의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { fileURLToPath } from "node:url";
import {
  GetDevicePositionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const locationClient = new LocationClient({ region: region });
  const deviceId = `${data.inputs.deviceId}`;
  const trackerName = `${data.inputs.trackerName}`;

  const devicePositionParams = {
    DeviceId: deviceId,
    TrackerName: trackerName,
  };
  try {
    const command = new GetDevicePositionCommand(devicePositionParams);
    const response = await locationClient.send(command);
    //state.position = response.position;
    console.log("Successfully fetched device position: ", response);
  } catch (error) {
    console.log("Error ", error);
    /* if (caught instanceof ResourceNotFoundException) {
      console.error(
        `The resource was not found: ${caught.message} \n Exiting program.`
      );
    } else {
```

```

    `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
  }
  return;*/
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetDevicePosition](#)을 참조하세요.

PutGeofence

다음 코드 예시는 PutGeofence의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { fileURLToPath } from "node:url";
import {
  PutGeofenceCommand,
  LocationClient,
  ValidationException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });
export const main = async () => {
  const geoFenceGeoParams = {
    CollectionName: `${data.inputs.collectionName}`,
    GeofenceId: `${data.inputs.geoId}`,
    Geometry: {
      Polygon: [
        [-122.3381, 47.6101],
        [-122.3281, 47.6101],

```

```

        [-122.3281, 47.6201],
        [-122.3381, 47.6201],
        [-122.3381, 47.6101],
    ],
],
},
};
try {
    const command = new PutGeofenceCommand(geoFenceGeoParams);
    const response = await locationClient.send(command);
    console.log("GeoFence created. GeoFence ID is: ", response.GeofenceId);
} catch (error) {
    console.error(
        `A validation error occurred while creating geofence: ${error} \n Exiting
program.`
    );
    return;
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutGeofence](#)를 참조하세요.

SDK for JavaScript(v3)를 사용한 Amazon MSK 예제

다음 코드 예제에서는 Amazon MSK에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [서버리스 예제](#)

서버리스 예제

Amazon MSK 트리거를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon MSK 클러스터에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 MSK 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 Amazon MSK 이벤트를 사용합니다.

```
exports.handler = async (event) => {
  // Iterate through keys
  for (let key in event.records) {
    console.log('Key: ', key)
    // Iterate through records
    event.records[key].map((record) => {
      console.log('Record: ', record)
      // Decode base64
      const msg = Buffer.from(record.value, 'base64').toString()
      console.log('Message:', msg)
    })
  }
}
```

TypeScript를 사용하여 Lambda로 Amazon MSK 이벤트를 사용합니다.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
```



```
    serviceName: "msk-handler-sample",
  });

export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    }
  }
}
```

SDK for JavaScript (v3)를 사용한 Amazon Personalize 예

다음 코드 예제에서는 Personalize에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

CreateBatchInferenceJob

다음 코드 예시는 CreateBatchInferenceJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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: {
    s3DataSource: {
      path: "INPUT_PATH",
    },
  },
  jobOutput: {
    s3DataDestination: {
      path: "OUTPUT_PATH",
    },
  },
  roleArn: "ROLE_ARN",
  solutionVersionArn: "SOLUTION_VERSION_ARN",
  numResults: 20,
};
```

```
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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateBatchInferenceJob](#)을 참조하세요.

CreateBatchSegmentJob

다음 코드 예시는 CreateBatchSegmentJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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: {
      s3DataSource: {
        path: "INPUT_PATH",
      },
    },
    jobOutput: {
      s3DataDestination: {
        path: "OUTPUT_PATH",
      },
    },
    roleArn: "ROLE_ARN",
    solutionVersionArn: "SOLUTION_VERSION_ARN",
    numResults: 20,
  };

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 참조의 [CreateBatchSegmentJob](#)을 참조하세요.

CreateCampaign

다음 코드 예시는 CreateCampaign의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateCampaign](#)을 참조하세요.

CreateDataset

다음 코드 예시는 CreateDataset의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateDataset](#)를 참조하세요.

CreateDatasetExportJob

다음 코드 예시는 CreateDatasetExportJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateDatasetExportJob](#)을 참조하세요.

CreateDatasetGroup

다음 코드 예시는 CreateDatasetGroup의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 dsd, 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateDatasetGroup](#)을 참조하세요.

CreateDatasetImportJob

다음 코드 예시는 CreateDatasetImportJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateDatasetImportJob](#)을 참조하세요.

CreateEventTracker

다음 코드 예시는 CreateEventTracker의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateEventTracker](#)를 참조하세요.

CreateFilter

다음 코드 예시는 CreateFilter의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateFilter](#)를 참조하세요.

CreateRecommender

다음 코드 예시는 CreateRecommender의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateRecommender](#)를 참조하세요.

CreateSchema

다음 코드 예시는 CreateSchema의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 "node:fs";

const 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();
```

도메인 포함 스키마 생성

```
// 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 "node:fs";

const 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateSchema](#)를 참조하세요.

CreateSolution

다음 코드 예시는 CreateSolution의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateSolution](#)을 참조하세요.

CreateSolutionVersion

다음 코드 예시는 CreateSolutionVersion의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateSolutionVersion](#)을 참조하세요.

SDK for JavaScript (v3)를 사용한 Amazon Personalize 이벤트 예

다음 코드 예제에서는 Personalize Events와 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

PutEvents

다음 코드 예시는 PutEvents의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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.
const 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 참조의 [PutEvents](#)를 참조하세요.

PutItems

다음 코드 예시는 PutItems의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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.
const 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 참조의 [PutItems](#)를 참조하세요.

PutUsers

다음 코드 예시는 PutUsers의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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.
const 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 참조의 [PutUsers](#)를 참조하세요.

SDK for JavaScript (v3)를 사용한 Amazon Personalize 런타임 예

다음 코드 예제에서는 Personalize 런타임과 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

GetPersonalizedRanking

다음 코드 예시는 GetPersonalizedRanking의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [GetPersonalizedRanking](#)을 참조하세요.

GetRecommendations

다음 코드 예시는 GetRecommendations의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetRecommendations](#)를 참조하세요.

SDK for JavaScript (v3)를 사용한 Amazon Pinpoint 예제

다음 코드 예제에서는 Amazon Pinpoint에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

SendMessage

다음 코드 예시는 SendMessage의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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";
export const pinClient = new PinpointClient({ region: REGION });
```

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.
const subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
const 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.
const body_html = `
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint 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.
const 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 { MessageResponse } = await pinClient.send(
      new SendMessagesCommand(params),
    );

    if (!MessageResponse) {
      throw new Error("No message response.");
    }

    if (!MessageResponse.Result) {
      throw new Error("No message result.");
    }
  }
}
```

```

    const recipientResult = MessageResponse.Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
        throw new Error(recipientResult.StatusMessage);
    }
    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";

/* The phone number or short code to send the message from. The phone number
   or short code that you specify has to be associated with your Amazon Pinpoint
   account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number. For best results, you should specify the phone
   number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
    "This message was sent through Amazon Pinpoint " +
    "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
    "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
   Make sure that the SMS channel is enabled for the project or application
   that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
   time-sensitive content, specify TRANSACTIONAL. If you plan to send

```

```
marketing-related content, specify PROMOTIONAL.*/
const messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
const 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.*/

const senderId = "MySenderId";

// Specify the parameters to pass to the API.
const 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));
    console.log(
      `Message sent!
      ${data.MessageResponse.Result[destinationNumber].StatusMessage}`,
    );
  } catch (err) {
    console.log(err);
  }
};
```

```
run();
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [SendMessages](#)를 참조하세요.

SDK for JavaScript(v3)를 사용한 Amazon Polly 예제

다음 코드 예제에서는 Amazon Polly에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

고객 피드백 분석을 위한 애플리케이션 생성

다음 코드 예제에서는 고객 의견 카드를 분석하고, 원어에서 번역하고, 감정을 파악하고, 번역된 텍스트에서 오디오 파일을 생성하는 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 코드와 배포 지침은 [GitHub](#)의 프로젝트를 참조하세요. 다음 발췌문은 Lambda 함수 내에서 AWS SDK for JavaScript가 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 예제에서 사용되는 서비스

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

SDK for JavaScript(v3)를 사용한 Amazon RDS 예제

다음 코드 예제에서는 Amazon RDS에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)
- [서버리스 예제](#)

시나리오

Aurora 서버리스 작업 항목 트래커 만들기

다음 코드 예제에서는 Amazon Aurora Serverless 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 사용하여 보고서를 보내는 웹 애플리케이션을 만드는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 보여줍니다. 이 예제에서는 Express Node.js 백엔드와의 상호 작용을 위해 React.js로 빌드된 프론트엔드를 사용합니다.

- React.js 웹 애플리케이션을와 통합합니다 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

서버리스 예제

Lambda 함수를 사용하여 Amazon RDS 데이터베이스에 연결

다음 코드 예제는 RDS 데이터베이스에 연결하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 간단한 데이터베이스 요청을 하고 결과를 반환합니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda 함수에서 Amazon RDS 데이터베이스에 연결

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
```

```

let connectionConfig = {
  host: process.env.ProxyHostName,
  user: process.env.DBUserName,
  password: token,
  database: process.env.DBName,
  ssl: 'Amazon RDS'
}
// Create the connection to the DB
const conn = await mysql.createConnection(connectionConfig);
// Obtain the result of the query
const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
return res;
}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};

```

TypeScript를 사용하여 Lambda 함수에서 Amazon RDS 데이터베이스에 연결

```

import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
// DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!

async function createAuthToken(): Promise<string> {

```

```
// Create RDS Signer object
const signer = new Signer({
  hostname: proxy_host_name,
  port: port,
  region: aws_region,
  username: db_user_name
});

// Request authorization token from RDS, specifying the username
const token = await signer.getAuthToken();
return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
  try {
    // Obtain auth token
    const token = await createAuthToken();
    const conn = await mysql.createConnection({
      host: proxy_host_name,
      user: db_user_name,
      password: token,
      database: db_name,
      ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
    });
    const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
    console.log('result:', rows);
    return rows;
  }
  catch (err) {
    console.log(err);
  }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
string }> => {
  // Execute database flow
  const result = await dbOps();

  // Return error is result is undefined
  if (result == undefined)
    return {
      statusCode: 500,
      body: JSON.stringify('Error with connection to DB host')
    }
}
```

```
    }

    // Return result
    return {
      statusCode: 200,
      body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

SDK for JavaScript(v3)를 사용한 Amazon RDS 데이터 서비스 예제

다음 코드 예제에서는 Amazon RDS Data Service에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

Aurora 서버리스 작업 항목 트래커 만들기

다음 코드 예제에서는 Amazon Aurora Serverless 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 사용하여 보고서를 보내는 웹 애플리케이션을 만드는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 보여줍니다. 이 예제에서는 Express Node.js 백엔드와의 상호 작용을 위해 React.js로 빌드된 프론트엔드를 사용합니다.

- React.js 웹 애플리케이션을와 통합합니다 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

SDK for JavaScript (v3)를 사용한 Amazon Redshift 예

다음 코드 예제에서는 Amazon Redshift에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

CreateCluster

다음 코드 예시는 CreateCluster의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
import { RedshiftClient } from "@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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateCluster](#)를 참조하세요.

DeleteCluster

다음 코드 예시는 DeleteCluster의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
import { RedshiftClient } from "@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 참조의 [DeleteCluster](#)를 참조하세요.

DescribeClusters

다음 코드 예시는 DescribeClusters의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
import { RedshiftClient } from "@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 참조의 [DescribeClusters](#)를 참조하세요.

ModifyCluster

다음 코드 예시는 ModifyCluster의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
import { RedshiftClient } from "@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();
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [ModifyCluster](#)를 참조하세요.

SDK for JavaScript(v3)를 사용한 Amazon Rekognition 예제

다음 코드 예제에서는 Amazon Rekognition에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

사진을 관리하기 위한 서버리스 애플리케이션 만들기

다음 코드 예시에서는 사용자가 레이블을 사용하여 사진을 관리할 수 있는 서버리스 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 [AWS 커뮤니티](#)의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

이미지에서 객체 감지

다음 코드 예제에서는 Amazon Rekognition을 사용하여 이미지에서 범주별 객체를 감지하는 앱을 구축하는 방법을 보여줍니다.

SDK for JavaScript(v3)

에서 Amazon Rekognition AWS SDK for JavaScript 을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 이미지에서 Amazon Rekognition을 사용하여 범주별로 객체를

식별하는 앱을 생성하는 방법을 보여줍니다. 이 앱은 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 만듭니다.
- 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

SDK for JavaScript (v3)를 사용한 Amazon S3 예제

다음 코드 예제에서는 Amazon S3에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)
- [시나리오](#)

- [서버리스 예제](#)

시작하기

Hello Amazon S3

다음 코드 예제에서는 Amazon S3 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  paginateListBuckets,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * List the S3 buckets in your configured AWS account.
 */
export const helloS3 = async () => {
  // When no region or credentials are provided, the SDK will use the
  // region and credentials from the local AWS config.
  const client = new S3Client({});

  try {
    /**
     * @type { import("@aws-sdk/client-s3").Bucket[] }
     */
    const buckets = [];

    for await (const page of paginateListBuckets({ client }, {})) {
      buckets.push(...page.Buckets);
    }
    console.log("Buckets: ");
    console.log(buckets.map((bucket) => bucket.Name).join("\n"));
    return buckets;
  }
}
```

```

    } catch (caught) {
      // ListBuckets does not throw any modeled errors. Any error caught
      // here will be something generic like `AccessDenied`.
      if (caught instanceof S3ServiceException) {
        console.error(`${caught.name}: ${caught.message}`);
      } else {
        // Something besides S3 failed.
        throw caught;
      }
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListBuckets](#)를 참조하세요.

기본 사항

기본 사항 알아보기

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.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

먼저 필요한 모듈을 모두 가져옵니다.

```

// Used to check if currently running file is this file.
import { fileURLToPath } from "node:url";

```

```
import { readdirSync, readFileSync, writeFileSync } from "node: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 연결된 GitHub 리포지토리에 로컬로 제공됩니다. 참조를 위해 해당 유틸리티의 다음 구현을 참조하세요.

```
export const dirnameFromMetaUrl = (metaUrl) =>
  fileURLToPath(new URL(".", metaUrl));

import { select, input, confirm, checkbox, password } 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 {{ message: string }} options
 */
password(options) {
  return password({ ...options, mask: true });
}

/**
 * @param {string} prompt
 */
checkContinue = async (prompt = "") => {
  const prefix = prompt && `${prompt} `;
  const ok = await this.confirm({
    message: `${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`;
};

```

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 (const 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 확인하세요. 이를 위해 ListObjects를 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. 이를 위해서는 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;
  }
  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 (const 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 항목을 참조하세요.

- [CopyObject](#)
- [CreateBucket](#)
- [DeleteBucket](#)
- [DeleteObjects](#)
- [GetObject](#)
- [ListObjectsV2](#)
- [PutObject](#)

작업

CopyObject

다음 코드 예시는 CopyObject의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

객체를 복사합니다.

```
import {  
  S3Client,  
  CopyObjectCommand,  
  ObjectNotInActiveTierError,
```

```

    waitUntilObjectExists,
  } from "@aws-sdk/client-s3";

/**
 * Copy an S3 object from one bucket to another.
 *
 * @param {{
 *   sourceBucket: string,
 *   sourceKey: string,
 *   destinationBucket: string,
 *   destinationKey: string }} config
 */
export const main = async ({
  sourceBucket,
  sourceKey,
  destinationBucket,
  destinationKey,
}) => {
  const client = new S3Client({});

  try {
    await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucket}/${sourceKey}`,
        Bucket: destinationBucket,
        Key: destinationKey,
      }),
    );
    await waitUntilObjectExists(
      { client },
      { Bucket: destinationBucket, Key: destinationKey },
    );
    console.log(
      `Successfully copied ${sourceBucket}/${sourceKey} to ${destinationBucket}/${destinationKey}`,
    );
  } catch (caught) {
    if (caught instanceof ObjectNotInActiveTierError) {
      console.error(
        `Could not copy ${sourceKey} from ${sourceBucket}. Object is not in the active tier.`,
      );
    } else {
      throw caught;
    }
  }
};

```

```

    }
  }
};

```

객체의 ETag가 제공된 값과 일치하지 않을 경우에만 조건부로 객체를 복사합니다.

```

import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
  string, eTag: string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
  eTag,
}) => {
  const client = new S3Client({});
  const name = data.name;
  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucketName}/${sourceKeyName}`,
        Bucket: destinationBucketName,
        Key: `${name}${sourceKeyName}`,
        CopySourceIfMatch: eTag,
      }),
    );
    console.log("Successfully copied object to bucket.");
  }

```

```
    } catch (caught) {
      if (caught instanceof NoSuchKey) {
        console.error(
          `Error from S3 while copying object "${sourceKeyName}" from
"${sourceBucketName}". No such key exists.`,
        );
      } else if (caught instanceof S3ServiceException) {
        console.error(
          `Unable to copy object "${sourceKeyName}" to bucket "${sourceBucketName}":
${caught.name}: ${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
```

```

    const { errors } = validateArgs({ options }, results);
    return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}

```

객체의 ETag가 제공된 값과 일치하지 않을 경우에만 조건부로 객체를 복사합니다.

```

import {
    CopyObjectCommand,
    NoSuchKey,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
    type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 * string, eTag: string }}
 */
export const main = async ({
    sourceBucketName,
    sourceKeyName,
    destinationBucketName,
    eTag,
}) => {
    const client = new S3Client({});
    const name = data.name;

```

```

try {
  const response = await client.send(
    new CopyObjectCommand({
      CopySource: `${sourceBucketName}/${sourceKeyName}`,
      Bucket: destinationBucketName,
      Key: `${name}${sourceKeyName}`,
      CopySourceIfNoneMatch: eTag,
    }),
  );
  console.log("Successfully copied object to bucket.");
} catch (caught) {
  if (caught instanceof NoSuchKey) {
    console.error(
      `Error from S3 while copying object "${sourceKeyName}" from
"${sourceBucketName}". No such key exists.`,
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Unable to copy object "${sourceKeyName}" to bucket "${sourceBucketName}":
${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
  },

```

```

    destinationBucketName: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

지정된 기간에 생성되거나 수정된 객체만 조건부로 복사합니다.

```

import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
  string }}

```

```
*/
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
}) => {
  const date = new Date();
  date.setDate(date.getDate() - 1);

  const name = data.name;
  const client = new S3Client({});
  const copySource = `${sourceBucketName}/${sourceKeyName}`;
  const copiedKey = name + sourceKeyName;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: copySource,
        Bucket: destinationBucketName,
        Key: copiedKey,
        CopySourceIfModifiedSince: date,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
"${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while copying object from ${sourceBucketName}.
${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
```



```

    isMain,
    validateArgs,
  } from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

지정된 기간에 생성되거나 수정되지 않은 객체만 조건부로 복사합니다.

```

import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

```

```
// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
  string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
}) => {
  const date = new Date();
  date.setDate(date.getDate() - 1);
  const client = new S3Client({});
  const name = data.name;
  const copiedKey = name + sourceKeyName;
  const copySource = `${sourceBucketName}/${sourceKeyName}`;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: copySource,
        Bucket: destinationBucketName,
        Key: copiedKey,
        CopySourceIfUnmodifiedSince: date,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
        "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while copying object from ${sourceBucketName}.
        ${caught.name}: ${caught.message}`,
      );
    }
  }
}
```

```
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CopyObject](#)를 참조하세요.

CreateBucket

다음 코드 예시는 CreateBucket의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷을 생성합니다.

```
import {
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  CreateBucketCommand,
  S3Client,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

/**
 * Create an Amazon S3 bucket.
 * @param {{ bucketName: string }} config
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { Location } = await client.send(
      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: bucketName,
      })
    );
    await waitUntilBucketExists({ client }, { Bucket: bucketName });
    console.log(`Bucket created with location ${Location}`);
  } catch (caught) {
    if (caught instanceof BucketAlreadyExists) {
```

```

    console.error(
      `The bucket "${bucketName}" already exists in another AWS account. Bucket
names must be globally unique.`,
    );
  }
  // WARNING: If you try to create a bucket in the North Virginia region,
  // and you already own a bucket in that region with the same name, this
  // error will not be thrown. Instead, the call will return successfully
  // and the ACL on that bucket will be reset.
  else if (caught instanceof BucketAlreadyOwnedByYou) {
    console.error(
      `The bucket "${bucketName}" already exists in this AWS account.`,
    );
  } else {
    throw caught;
  }
}
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateBucket](#)을 참조하세요.

DeleteBucket

다음 코드 예시는 DeleteBucket의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷을 삭제합니다.

```

import {
  DeleteBucketCommand,
  S3Client,
  S3ServiceException,

```

```

} from "@aws-sdk/client-s3";

/**
 * Delete an Amazon S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new DeleteBucketCommand({
    Bucket: bucketName,
  });

  try {
    await client.send(command);
    console.log("Bucket was deleted.");
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting bucket. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting the bucket. ${caught.name}:`,
        `${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteBucket](#)을 참조하세요.

DeleteBucketPolicy

다음 코드 예시는 DeleteBucketPolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷 정책을 삭제합니다.

```
import {
  DeleteBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Remove the policy from an Amazon S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteBucketPolicyCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`Bucket policy deleted from "${bucketName}".`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting policy from ${bucketName}. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting policy from ${bucketName}. ${caught.name}: ${caught.message}`,
      );
    }
  }
}
```

```

    );
  } else {
    throw caught;
  }
}
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteBucketPolicy](#)를 참조하세요.

DeleteBucketWebsite

다음 코드 예시는 DeleteBucketWebsite의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷에서 웹 사이트 구성을 삭제합니다.

```

import {
  DeleteBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Remove the website configuration for a bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteBucketWebsiteCommand({

```



```
        Bucket: bucketName,
      })),
    );
    // The response code will be successful for both removed configurations and
    // configurations that did not exist in the first place.
    console.log(
      `The bucket "${bucketName}" is not longer configured as a website, or it never
was.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while removing website configuration from ${bucketName}. The
bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while removing website configuration from ${bucketName}.
${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteBucketWebsite](#)를 참조하세요.

DeleteObject

다음 코드 예시는 DeleteObject의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

객체를 삭제합니다.

```
import {
  DeleteObjectCommand,
  S3Client,
  S3ServiceException,
  waitUntilObjectNotExists,
} from "@aws-sdk/client-s3";

/**
 * Delete one object from an Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteObjectCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    await waitUntilObjectNotExists(
      { client },
      { Bucket: bucketName, Key: key },
    );
    // A successful delete, or a delete for a non-existent object, both return
    // a 204 response code.
    console.log(
      `The object "${key}" from bucket "${bucketName}" was deleted, or it didn't
      exist.`,
    );
  } catch (caught) {
    if (
```

```

        caught instanceof S3ServiceException &&
        caught.name === "NoSuchBucket"
    ) {
        console.error(
            `Error from S3 while deleting object from ${bucketName}. The bucket doesn't
            exist.` ,
        );
    } else if (caught instanceof S3ServiceException) {
        console.error(
            `Error from S3 while deleting object from ${bucketName}.  ${caught.name}:
            ${caught.message}` ,
        );
    } else {
        throw caught;
    }
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteObject](#)를 참조하세요.

DeleteObjects

다음 코드 예시는 DeleteObjects의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

여러 객체를 삭제합니다.

```

import {
    DeleteObjectsCommand,
    S3Client,
    S3ServiceException,
    waitUntilObjectNotExists,
} from "@aws-sdk/client-s3";

```

```

/**
 * Delete multiple objects from an S3 bucket.
 * @param {{ bucketName: string, keys: string[] }}
 */
export const main = async ({ bucketName, keys }) => {
  const client = new S3Client({});

  try {
    const { Deleted } = await client.send(
      new DeleteObjectsCommand({
        Bucket: bucketName,
        Delete: {
          Objects: keys.map((k) => ({ Key: k })),
        },
      }),
    );
    for (const key in keys) {
      await waitUntilObjectNotExists(
        { client },
        { Bucket: bucketName, Key: key },
      );
    }
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted
objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting objects from ${bucketName}. The bucket doesn't
exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting objects from ${bucketName}. ${caught.name}:
${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
}

```

```

    }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteObjects](#)를 참조하세요.

GetBucketAcl

다음 코드 예시는 GetBucketAcl의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

ACL 권한을 가져옵니다.

```

import {
  GetBucketAclCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Retrieves the Access Control List (ACL) for an S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetBucketAclCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`ACL for bucket "${bucketName}":`);
    console.log(JSON.stringify(response, null, 2));
  } catch (caught) {

```

```

    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting ACL for ${bucketName}. The bucket doesn't
        exist.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting ACL for ${bucketName}. ${caught.name}:
        ${caught.message}`
      );
    } else {
      throw caught;
    }
  }
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetBucketAcl](#)을 참조하세요.

GetBucketCors

다음 코드 예시는 GetBucketCors의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷에 대한 CORS 정책을 가져옵니다.

```

import {
  GetBucketCorsCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

```

```
/**
 * Log the Cross-Origin Resource Sharing (CORS) configuration information
 * set for the bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new GetBucketCorsCommand({
    Bucket: bucketName,
  });

  try {
    const { CORSRules } = await client.send(command);
    console.log(JSON.stringify(CORSRules));
    CORSRules.forEach((cr, i) => {
      console.log(
        `\\nCORSRule ${i + 1}`,
        `\\n${"-"}.repeat(10)`,
        `\\nAllowedHeaders: ${cr.AllowedHeaders}`,
        `\\nAllowedMethods: ${cr.AllowedMethods}`,
        `\\nAllowedOrigins: ${cr.AllowedOrigins}`,
        `\\nExposeHeaders: ${cr.ExposeHeaders}`,
        `\\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting bucket CORS rules for ${bucketName}. The bucket
        doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting bucket CORS rules for ${bucketName}.
        ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
}
```

```
};
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetBucketCors](#)를 참조하세요.

GetBucketPolicy

다음 코드 예시는 GetBucketPolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷 정책을 가져옵니다.

```
import {
  GetBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Logs the policy for a specified bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { Policy } = await client.send(
      new GetBucketPolicyCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`Policy for "${bucketName}":\n${Policy}`);
  } catch (caught) {
    if (
```



```

        caught instanceof S3ServiceException &&
        caught.name === "NoSuchBucket"
    ) {
        console.error(
            `Error from S3 while getting policy from ${bucketName}. The bucket doesn't
            exist.` ,
        );
    } else if (caught instanceof S3ServiceException) {
        console.error(
            `Error from S3 while getting policy from ${bucketName}.  ${caught.name}:
            ${caught.message}` ,
        );
    } else {
        throw caught;
    }
}
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetBucketPolicy](#)를 참조하세요.

GetBucketWebsite

다음 코드 예시는 GetBucketWebsite의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

웹 사이트 구성을 가져옵니다.

```

import {
    GetBucketWebsiteCommand,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";

```

```

/**
 * Log the website configuration for a bucket.
 * @param {{ bucketName }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetBucketWebsiteCommand({
        Bucket: bucketName,
      }),
    );
    console.log(
      `Your bucket is set up to host a website with the following configuration:\n
      ${JSON.stringify(response, null, 2)}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchWebsiteConfiguration"
    ) {
      console.error(
        `Error from S3 while getting website configuration for ${bucketName}. The
        bucket isn't configured as a website.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting website configuration for ${bucketName}.
        ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetBucketWebsite](#)를 참조하세요.

GetObject

다음 코드 예시는 GetObject의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

객체를 다운로드합니다.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
        key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
```

```

        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
        ${caught.message}`,
        );
    } else {
        throw caught;
    }
}
};

```

객체의 ETag가 제공된 값과 일치하는 경우에만 조건부로 객체를 다운로드합니다.

```

import {
    GetObjectCommand,
    NoSuchKey,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
    const client = new S3Client({});

    try {
        const response = await client.send(
            new GetObjectCommand({
                Bucket: bucketName,
                Key: key,
                IfMatch: eTag,
            }),
        );
        // The Body object also has 'transformToByteArray' and 'transformToWebStream'
        methods.
        const str = await response.Body.transformToString();
        console.log("Success. Here is text of the file:", str);
    } catch (caught) {
        if (caught instanceof NoSuchKey) {
            console.error(

```

```

        `Error from S3 while getting object "${key}" from "${bucketName}". No such
        key exists.` ,
    );
    } else if (caught instanceof S3ServiceException) {
        console.error(
            `Error from S3 while getting object from ${bucketName}. ${caught.name}:
            ${caught.message}` ,
        );
    } else {
        throw caught;
    }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
    isMain,
    validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
    const options = {
        bucketName: {
            type: "string",
            required: true,
        },
        key: {
            type: "string",
            required: true,
        },
        eTag: {
            type: "string",
            required: true,
        },
    };
};
const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {

```

```

    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

객체의 ETag가 제공된 값과 일치하지 않을 경우에만 조건부로 객체를 다운로드합니다.

```

import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfNoneMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
        key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {

```

```
        console.error(
            `Error from S3 while getting object from ${bucketName}. ${caught.name}:
${caught.message}`,
        );
    } else {
        throw caught;
    }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
    isMain,
    validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
    const options = {
        bucketName: {
            type: "string",
            required: true,
        },
        key: {
            type: "string",
            required: true,
        },
        eTag: {
            type: "string",
            required: true,
        },
    };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}
```

```
}
```

지정된 기간에 생성되거나 수정된 객체만 조건부로 다운로드합니다.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const date = new Date();
  date.setDate(date.getDate() - 1);
  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfModifiedSince: date,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
        key exists.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}. ${caught.name}:
        ${caught.message}`
      );
    }
  }
}
```



```
    );
  } else {
    throw caught;
  }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

지정된 기간에 생성되거나 수정되지 않은 객체만 조건부로 다운로드합니다.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const date = new Date();
  date.setDate(date.getDate() - 1);
  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfUnmodifiedSince: date,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}. ${caught.name}:
${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

```
// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetObject](#)를 참조하세요.

GetObjectLegalHold

다음 코드 예시는 GetObjectLegalHold의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  GetObjectLegalHoldCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get an object's current legal hold status.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectLegalHoldCommand({
        Bucket: bucketName,
        Key: key,
        // Optionally, you can provide additional parameters
        // ExpectedBucketOwner: "<account ID that is expected to own the bucket>",
        // VersionId: "<the specific version id of the object to check>",
      }),
    );
    console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting legal hold status for ${key} in ${bucketName}. The bucket doesn't exist.`
      );
    }
  }
}
```

```
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting legal hold status for ${key} in ${bucketName}
from ${bucketName}.  ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetObjectLegalHold](#)를 참조하세요.

GetObjectLockConfiguration

다음 코드 예시는 GetObjectLockConfiguration의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  GetObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Gets the Object Lock configuration for a bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { ObjectLockConfiguration } = await client.send(
      new GetObjectLockConfigurationCommand({
        Bucket: bucketName,
        // Optionally, you can provide additional parameters
        // ExpectedBucketOwner: "<account ID that is expected to own the bucket>",
      }),
    );
    console.log(
      `Object Lock Configuration:\n${JSON.stringify(ObjectLockConfiguration)}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    )
```

```
    ) {
      console.error(
        `Error from S3 while getting object lock configuration for ${bucketName}.
        The bucket doesn't exist.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object lock configuration for ${bucketName}.
        ${caught.name}: ${caught.message}`
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetObjectLockConfiguration](#)을 참조하세요.

GetObjectRetention

다음 코드 예시는 GetObjectRetention의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  GetObjectRetentionCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the "RetainUntilDate" for an object in an S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const { Retention } = await client.send(
      new GetObjectRetentionCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    console.log(
      `${key} in ${bucketName} will be retained until ${Retention.RetainUntilDate}`,
    );
  }
}
```



```
    } catch (caught) {
      if (
        caught instanceof S3ServiceException &&
        caught.name === "NoSuchObjectLockConfiguration"
      ) {
        console.warn(
          `The object "${key}" in the bucket "${bucketName}" does not have an
ObjectLock configuration.`
        );
      } else if (caught instanceof S3ServiceException) {
        console.error(
          `Error from S3 while getting object retention settings for "${bucketName}".
${caught.name}: ${caught.message}`
        );
      } else {
        throw caught;
      }
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
```

```
const { errors, results } = loadArgs();
if (!errors) {
  main(results.values);
} else {
  console.error(errors.join("\n"));
}
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetObjectRetention](#)을 참조하세요.

ListBuckets

다음 코드 예시는 ListBuckets의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷을 나열합니다.

```
import {
  paginateListBuckets,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * List the Amazon S3 buckets in your account.
 */
export const main = async () => {
  const client = new S3Client({});
  /** @type {?import('@aws-sdk/client-s3').Owner} */
  let Owner = null;

  /** @type {import('@aws-sdk/client-s3').Bucket[]} */
  const Buckets = [];
```

```

try {
  const paginator = paginateListBuckets({ client }, {});

  for await (const page of paginator) {
    if (!Owner) {
      Owner = page.Owner;
    }

    Buckets.push(...page.Buckets);
  }

  console.log(
    `${Owner.DisplayName} owns ${Buckets.length} bucket${
      Buckets.length === 1 ? "" : "s"
    }:`,
  );
  console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
} catch (caught) {
  if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while listing buckets.  ${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListBuckets](#)를 참조하세요.

ListObjectsV2

다음 코드 예시는 ListObjectsV2의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷의 모든 객체를 나열합니다. 객체가 두 개 이상인 경우, 전체 목록을 반복하는 데 `IsTruncated` 및 `NextContinuationToken`이 사용됩니다.

```
import {
  S3Client,
  S3ServiceException,
  // This command supersedes the ListObjectsCommand and is the recommended way to
  list objects.
  paginateListObjectsV2,
} from "@aws-sdk/client-s3";

/**
 * Log all of the object keys in a bucket.
 * @param {{ bucketName: string, pageSize: string }}
 */
export const main = async ({ bucketName, pageSize }) => {
  const client = new S3Client({});
  /** @type {string[][]} */
  const objects = [];
  try {
    const paginator = paginateListObjectsV2(
      { client, /* Max items per page */ pageSize: Number.parseInt(pageSize) },
      { Bucket: bucketName },
    );

    for await (const page of paginator) {
      objects.push(page.Contents.map((o) => o.Key));
    }
    objects.forEach((objectList, pageNum) => {
      console.log(
        `Page ${pageNum + 1}\n-----\n${objectList.map((o) => `•
${o}`)}.join("\n")\n`,
      );
    });
  }
};
```

```

    } catch (caught) {
      if (
        caught instanceof S3ServiceException &&
        caught.name === "NoSuchBucket"
      ) {
        console.error(
          `Error from S3 while listing objects for "${bucketName}". The bucket doesn't
exist.`,
        );
      } else if (caught instanceof S3ServiceException) {
        console.error(
          `Error from S3 while listing objects for "${bucketName}". ${caught.name}:
${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  };

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListObjectsV2](#)를 참조하세요.

PutBucketAcl

다음 코드 예시는 PutBucketAcl의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷 ACL을 적용합니다.

```

import {
  PutBucketAclCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

```

```

/**
 * Grant read access to a user using their canonical AWS account ID.
 *
 * 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.
 * @param {{ bucketName: string, granteeCanonicalUserId: string,
 * ownerCanonicalUserId }}
 */
export const main = async ({
  bucketName,
  granteeCanonicalUserId,
  ownerCanonicalUserId,
}) => {
  const client = new S3Client({});
  const command = new PutBucketAclCommand({
    Bucket: bucketName,
    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: granteeCanonicalUserId,
            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: "READ",
        },
      ],
      Owner: {
        ID: ownerCanonicalUserId,
      },
    },
  });
};

```

```

try {
  await client.send(command);
  console.log(`Granted READ access to ${bucketName}`);
} catch (caught) {
  if (
    caught instanceof S3ServiceException &&
    caught.name === "NoSuchBucket"
  ) {
    console.error(
      `Error from S3 while setting ACL for bucket ${bucketName}. The bucket
doesn't exist.`
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while setting ACL for bucket ${bucketName}. ${caught.name}:
${caught.message}`
    );
  } else {
    throw caught;
  }
}
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutBucketAcl](#)을 참조하세요.

PutBucketCors

다음 코드 예시는 PutBucketCors의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CORS 규칙을 추가합니다.

```

import {
  PutBucketCorsCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Allows cross-origin requests to an S3 bucket by setting the CORS configuration.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new PutBucketCorsCommand({
        Bucket: bucketName,
        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 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,
            },
          ],
        },
      })
    );
    console.log(`Successfully set CORS rules for bucket: ${bucketName}`);
  } catch (caught) {

```



```

    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while setting CORS rules for ${bucketName}. The bucket
        doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting CORS rules for ${bucketName}. ${caught.name}:
        ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutBucketCors](#)를 참조하세요.

PutBucketPolicy

다음 코드 예시는 PutBucketPolicy의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

정책을 추가합니다.

```

import {
  PutBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

```

```

/**
 * Grant an IAM role GetObject access to all of the objects
 * in the provided bucket.
 * @param {{ bucketName: string, iamRoleArn: string }}
 */
export const main = async ({ bucketName, iamRoleArn }) => {
  const client = new S3Client({});
  const command = new PutBucketPolicyCommand({
    // This is a resource-based policy. For more information on resource-based
    policies,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/
    access_policies.html#policies_resource-based.
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {
            AWS: iamRoleArn,
          },
          Action: "s3:GetObject",
          Resource: `arn:aws:s3:::${bucketName}/*`,
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: bucketName,
  });

  try {
    await client.send(command);
    console.log(
      `GetObject access to the bucket "${bucketName}" was granted to the provided
IAM role.`
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "MalformedPolicy"
    ) {
      console.error(
        `Error from S3 while setting the bucket policy for the bucket
"${bucketName}". The policy was malformed.`
      );
    }
  }
}

```

```

    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while setting the bucket policy for the bucket
"${bucketName}". ${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutBucketPolicy](#)를 참조하세요.

PutBucketWebsite

다음 코드 예시는 PutBucketWebsite의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

웹 사이트 구성을 설정합니다.

```

import {
  PutBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Configure an Amazon S3 bucket to serve a static website.
 * Website access must also be granted separately. For more information
 * on setting the permissions for website access, see
 * https://docs.aws.amazon.com/AmazonS3/latest/userguide/
WebsiteAccessPermissionsReqd.html.

```

```
*
* @param {{ bucketName: string }}
*/
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new PutBucketWebsiteCommand({
    Bucket: bucketName,
    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 when the request is
        // for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
    await client.send(command);
    console.log(
      `The bucket "${bucketName}" has been configured as a static website.`
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while configuring the bucket "${bucketName}" as a static
website. The bucket doesn't exist.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while configuring the bucket "${bucketName}" as a static
website. ${caught.name}: ${caught.message}`
      );
    } else {
      throw caught;
    }
  }
};
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutBucketWebsite](#)를 참조하세요.

PutObject

다음 코드 예시는 PutObject의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

객체를 업로드합니다.

```
import { readFile } from "node:fs/promises";

import {
  PutObjectCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Upload a file to an S3 bucket.
 * @param {{ bucketName: string, key: string, filePath: string }}
 */
export const main = async ({ bucketName, key, filePath }) => {
  const client = new S3Client({});
  const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: key,
    Body: await readFile(filePath),
  });

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

```

    console.log(response);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "EntityTooLarge"
    ) {
      console.error(
        `Error from S3 while uploading object to ${bucketName}. \
The object was too large. To upload objects larger than 5GB, use the S3 console \
(160GB max) \
or the multipart upload API (5TB max).`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while uploading object to ${bucketName}.  ${caught.name}: \
${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

객체의 ETag가 제공된 값과 일치하는 경우에만 조건부로 객체를 업로드합니다.

```

import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(

```

```
    new GetObjectCommand({
      Bucket: bucketName,
      Key: key,
      IfMatch: eTag,
    }),
  );
  // The Body object also has 'transformToByteArray' and 'transformToWebStream'
  methods.
  const str = await response.Body.transformToString();
  console.log("Success. Here is text of the file:", str);
} catch (caught) {
  if (caught instanceof NoSuchKey) {
    console.error(
      `Error from S3 while getting object "${key}" from "${bucketName}". No such
      key exists.`
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while getting object from ${bucketName}. ${caught.name}:
      ${caught.message}`
    );
  } else {
    throw caught;
  }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  },
```

```

    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하십시오.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutObject](#)를 참조하세요.

PutObjectLegalHold

다음 코드 예시는 PutObjectLegalHold의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import {
  PutObjectLegalHoldCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

```



```
/**
 * Apply a legal hold configuration to the specified object.
 * @param {{ bucketName: string, objectKey: string, legalHoldStatus: "ON" | "OFF" }}
 */
export const main = async ({ bucketName, objectKey, legalHoldStatus }) => {
  if (!["OFF", "ON"].includes(legalHoldStatus.toUpperCase())) {
    throw new Error(
      "Invalid parameter. legalHoldStatus must be 'ON' or 'OFF'.",
    );
  }

  const client = new S3Client({});
  const command = new PutObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    LegalHold: {
      // Set the status to 'ON' to place a legal hold on the object.
      // Set the status to 'OFF' to remove the legal hold.
      Status: legalHoldStatus,
    },
  });

  try {
    await client.send(command);
    console.log(
      `Legal hold status set to "${legalHoldStatus}" for "${objectKey}" in "${bucketName}"`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying legal hold status for "${objectKey}" in "${bucketName}". The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying legal hold status for "${objectKey}" in "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
}
```

```
    }  
  }  
};  
  
// Call function if run directly  
import { parseArgs } from "node:util";  
import {  
  isMain,  
  validateArgs,  
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";  
  
const loadArgs = () => {  
  const options = {  
    bucketName: {  
      type: "string",  
      required: true,  
    },  
    objectKey: {  
      type: "string",  
      required: true,  
    },  
    legalHoldStatus: {  
      type: "string",  
      default: "ON",  
    },  
  };  
};  
const results = parseArgs({ options });  
const { errors } = validateArgs({ options }, results);  
return { errors, results };  
};  
  
if (isMain(import.meta.url)) {  
  const { errors, results } = loadArgs();  
  if (!errors) {  
    main(results.values);  
  } else {  
    console.error(errors.join("\n"));  
  }  
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutObjectLegalHold](#)를 참조하세요.

PutObjectLockConfiguration

다음 코드 예시는 PutObjectLockConfiguration의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

버킷의 객체 잠금 구성을 설정합니다.

```
import {
  PutObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Enable S3 Object Lock for an Amazon S3 bucket.
 * After you enable Object Lock on a bucket, you can't
 * disable Object Lock or suspend versioning for that bucket.
 * @param {{ bucketName: string, enabled: boolean }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
  });

  try {
    await client.send(command);
    console.log(`Object Lock for "${bucketName}" enabled.`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    )
  }
```

```
    ) {
      console.error(
        `Error from S3 while modifying the object lock configuration for the bucket
"${bucketName}". The bucket doesn't exist.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying the object lock configuration for the bucket
"${bucketName}". ${caught.name}: ${caught.message}`
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

버킷의 기본 보존 기간을 설정합니다.

```
import {
  PutObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Change the default retention settings for an object in an Amazon S3 bucket.
 * @param {{ bucketName: string, retentionDays: string }}
 */
export const main = async ({ bucketName, retentionDays }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: bucketName,
        // The Object Lock configuration that you want to apply to the specified
        bucket.
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
          Rule: {
            // The default Object Lock retention mode and period that you want to
            apply
            // to new objects placed in the specified bucket. Bucket settings
            require
            // both a mode and a period. The period can be either Days or Years but
            // you must select one.
            DefaultRetention: {
              // In governance mode, users can't overwrite or delete an object
              version
              // or alter its lock settings unless they have special permissions.
              With
              // governance mode, you protect objects against being deleted by most
              users,
              // but you can still grant some users permission to alter the
              retention settings
              // or delete the objects if necessary.
            }
          }
        }
      })
    );
  } catch (err) {
    if (err instanceof S3ServiceException) {
      // Handle S3ServiceException
    }
  }
}
```

```

        Mode: "GOVERNANCE",
        Days: Number.parseInt(retentionDays),
      },
    },
 )),
);
console.log(
  `Set default retention mode to "GOVERNANCE" with a retention period of
${retentionDays} day(s).`,
);
} catch (caught) {
  if (
    caught instanceof S3ServiceException &&
    caught.name === "NoSuchBucket"
  ) {
    console.error(
      `Error from S3 while setting the default object retention for a bucket. The
bucket doesn't exist.`,
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while setting the default object retention for a bucket.
${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  },

```

```

    retentionDays: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutObjectLockConfiguration](#)을 참조하세요.

PutObjectRetention

다음 코드 예시는 PutObjectRetention의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import {
  PutObjectRetentionCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

```

```
/**
 * Place a 24-hour retention period on an object in an Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: key,
    BypassGovernanceRetention: false,
    Retention: {
      // In governance mode, users can't overwrite or delete an object version
      // or alter its lock settings unless they have special permissions. With
      // governance mode, you protect objects against being deleted by most users,
      // but you can still grant some users permission to alter the retention
      settings
        // or delete the objects if necessary.
        Mode: "GOVERNANCE",
        RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
  });

  try {
    await client.send(command);
    console.log("Object Retention settings updated.");
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying the governance mode and retention period on
an object. The bucket doesn't exist.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying the governance mode and retention period on
an object. ${caught.name}: ${caught.message}`
      );
    } else {
      throw caught;
    }
  }
};
```



```
// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [PutObjectRetention](#)을 참조하세요.

시나리오

미리 서명된 URL 생성

다음 코드 예제에서는 Amazon S3에 대해 미리 서명된 URL을 생성하고 객체를 업로드하는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

미리 서명된 URL을 생성하여 버킷에 객체를 업로드합니다.

```
import https from "node:https";

import { XMLParser } from "fast-xml-parser";
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 });
};
```

```

/**
 * Make a PUT request to the provided URL.
 *
 * @param {string} url
 * @param {string} data
 */
const 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", () => {
          const parser = new XMLParser();
          if (res.statusCode >= 200 && res.statusCode <= 299) {
            resolve(parser.parse(responseBody, true));
          } else {
            reject(parser.parse(responseBody, true));
          }
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
};

/**
 * Create two presigned urls for uploading an object to an S3 bucket.
 * The first presigned URL is created with credentials from the shared INI file
 * in the current environment. The second presigned URL is created using an
 * existing S3Client instance that has already been provided with credentials.
 * @param {{ bucketName: string, key: string, region: string }}
 */
export const main = async ({ bucketName, key, region }) => {
  try {

```

```

const noClientUrl = await createPresignedUrlWithoutClient({
  bucket: bucketName,
  key,
  region,
});

const clientUrl = await createPresignedUrlWithClient({
  bucket: bucketName,
  region,
  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 (caught) {
  if (caught instanceof Error && caught.name === "CredentialsProviderError") {
    console.error(
      `There was an error getting your credentials. Are your local credentials
configured?\n${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}
};

```

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 });
};

/**
 * Create two presigned urls for downloading an object from an S3 bucket.
 * The first presigned URL is created with credentials from the shared INI file
 * in the current environment. The second presigned URL is created using an
 * existing S3Client instance that has already been provided with credentials.
 * @param {{ bucketName: string, key: string, region: string }}
 */
export const main = async ({ bucketName, key, region }) => {
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      bucket: bucketName,
      region,
      key,
    });

    const clientUrl = await createPresignedUrlWithClient({
      bucket: bucketName,
      region,
      key,
    });

    console.log("Presigned URL without client");
  }
};

```

```

    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "CredentialsProviderError") {
      console.error(
        `There was an error getting your credentials. Are your local credentials configured?\n${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하십시오.

사진을 관리하기 위한 서버리스 애플리케이션 만들기

다음 코드 예시에서는 사용자가 레이블을 사용하여 사진을 관리할 수 있는 서버리스 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 [AWS 커뮤니티](#)의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3

- Amazon SNS

Amazon S3 객체를 나열하는 웹 페이지 생성

다음 코드 예제에서는 웹 페이지에 Amazon S3 객체를 나열하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

다음 코드는 AWS SDK를 호출하는 관련 React 구성 요소입니다. 이 구성 요소가 포함된 실행 가능한 애플리케이션 버전은 이전 GitHub 링크에서 찾을 수 있습니다.

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  type 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 AWS SDK for JavaScript API 참조의 [ListObjects](#)를 참조하세요.

Amazon Textract 탐색기 애플리케이션 생성

다음 코드 예제에서는 대화형 애플리케이션을 통해 Amazon Textract 출력을 탐색하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript 를 사용하여 Amazon Textract를 사용하여 문서 이미지에서 데이터를 추출하고 대화형 웹 페이지에 표시하는 React 애플리케이션을 빌드하는 방법을 보여줍니다. 이 예제는 웹 브라우저에서 실행되며 자격 증명을 위해 인증된 Amazon Cognito 자격 증명에 필요합니다. 이 애플리케이션은 스토리지로 Amazon Simple Storage Service(Amazon 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

버킷의 모든 객체 삭제

다음 코드 예제는 Amazon S3 버킷의 모든 객체를 삭제하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

주어진 Amazon S3 버킷의 모든 객체를 삭제합니다.

```
import {
  DeleteObjectsCommand,
  paginateListObjectsV2,
  S3Client,
} from "@aws-sdk/client-s3";
```

```
/**
 *
 * @param {{ bucketName: string }} config
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  try {
    console.log(`Deleting all objects in bucket: ${bucketName}`);

    const paginator = paginateListObjectsV2(
      { client },
      {
        Bucket: bucketName,
      },
    );

    const objectKeys = [];
    for await (const { Contents } of paginator) {
      objectKeys.push(...Contents.map((obj) => ({ Key: obj.Key })));
    }

    const deleteCommand = new DeleteObjectsCommand({
      Bucket: bucketName,
      Delete: { Objects: objectKeys },
    });

    await client.send(deleteCommand);

    console.log(`All objects deleted from bucket: ${bucketName}`);
  } catch (caught) {
    if (caught instanceof Error) {
      console.error(
        `Failed to empty ${bucketName}. ${caught.name}: ${caught.message}`,
      );
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    bucketName: {
```

```
    type: "string",
  },
};

const { values } = parseArgs({ options });
main(values);
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.

- [DeleteObjects](#)
- [ListObjectsV2](#)

이미지에서 객체 감지

다음 코드 예제에서는 Amazon Rekognition을 사용하여 이미지에서 범주별 객체를 감지하는 앱을 구축하는 방법을 보여줍니다.

SDK for JavaScript(v3)

와 함께 Amazon Rekognition AWS SDK for JavaScript 을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 이미지에서 Amazon Rekognition을 사용하여 범주별로 객체를 식별하는 앱을 생성하는 방법을 보여줍니다. 이 앱은 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 만듭니다.
- 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

Amazon S3 객체 잠그기

다음 코드 예시에는 S3 객체 잠금 기능을 사용하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

시나리오의 진입점입니다(index.js). 모든 단계를 오케스트레이션합니다. GitHub를 방문하여 Scenario, ScenarioInput, ScenarioOutput, ScenarioAction에 대한 구현 세부 정보를 확인하세요.

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "../welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  confirmSetLegalHoldFileEnabled,
  confirmSetLegalHoldFileRetention,
  confirmSetRetentionPeriodFileEnabled,
  confirmSetRetentionPeriodFileRetention,
  confirmUpdateLockPolicy,
  confirmUpdateRetention,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
  setLegalHoldFileEnabledAction,
  setLegalHoldFileRetentionAction,
  setRetentionPeriodFileEnabledAction,
  setRetentionPeriodFileRetentionAction,
  updateLockPolicy,
  updateLockPolicyAction,
```

```
    updateRetention,
    updateRetentionAction,
  } from "./setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
    deploy: new scenarios.Scenario(
      "S3 Object Locking - Deploy",
      [
        welcome(scenarios),
        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        getBucketPrefix(scenarios),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
        createBucketsAction(scenarios, client),
        updateRetention(scenarios),
        confirmUpdateRetention(scenarios),
        exitOnFalse(scenarios, "confirmUpdateRetention"),
        updateRetentionAction(scenarios, client),
        populateBuckets(scenarios),
        confirmPopulateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        updateLockPolicy(scenarios),
        confirmUpdateLockPolicy(scenarios),
        exitOnFalse(scenarios, "confirmUpdateLockPolicy"),
        updateLockPolicyAction(scenarios, client),
        confirmSetLegalHoldFileEnabled(scenarios),
        setLegalHoldFileEnabledAction(scenarios, client),
        confirmSetRetentionPeriodFileEnabled(scenarios),
        setRetentionPeriodFileEnabledAction(scenarios, client),
        confirmSetLegalHoldFileRetention(scenarios),
        setLegalHoldFileRetentionAction(scenarios, client),
        confirmSetRetentionPeriodFileRetention(scenarios),
        setRetentionPeriodFileRetentionAction(scenarios, client),
        saveState,
```

```

    ],
    initialState,
  ),
  demo: new scenarios.Scenario(
    "S3 Object Locking - Demo",
    [loadState, replAction(scenarios, client)],
    initialState,
  ),
  clean: new scenarios.Scenario(
    "S3 Object Locking - Destroy",
    [
      loadState,
      confirmCleanup(scenarios),
      exitOnFalse(scenarios, "confirmCleanup"),
      cleanupAction(scenarios, client),
    ],
    initialState,
  ),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios, {
    name: "Amazon S3 object locking workflow",
    description:
      "Work with Amazon Simple Storage Service (Amazon S3) object locking",
    features: "",
    synopsis:
      "node index.js --scenario <deploy | demo | clean> [-h|--help] [-y|--yes] [-v|--verbose]",
  });
}

```

환영 메시지를 콘솔에 출력합니다(welcome.steps.js).

```

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    "Welcome to the Amazon Simple Storage Service (S3) Object Locking Feature
    Scenario. For this workflow, we will use the AWS SDK for JavaScript to create
    several S3 buckets and files to demonstrate working with S3 locking features.",
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };

```

버킷, 객체, 파일 설정을 배포합니다(setup.steps.js).

```

import {
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,

```

```

    GetBucketVersioningCommand,
    BucketAlreadyExists,
    BucketAlreadyOwnedByYou,
    S3ServiceException,
    waitUntilBucketExists,
  } from "@aws-sdk/client-s3";

import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const getBucketPrefix = (scenarios) =>
  new scenarios.ScenarioInput(
    "bucketPrefix",
    "Provide a prefix that will be used for bucket creation.",
    { type: "input", default: "amzn-s3-demo-bucket" },
  );

/**
 * @param {Scenarios} scenarios
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    (state) => `The following buckets will be created:
      ${state.bucketPrefix}-no-lock with object lock False.
      ${state.bucketPrefix}-lock-enabled with object lock True.
      ${state.bucketPrefix}-retention-after-creation with object lock False.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>

```



```

    new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
      type: "confirm",
    });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) => {
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const noLockBucketName = `${state.bucketPrefix}-no-lock`;
    const lockEnabledBucketName = `${state.bucketPrefix}-lock-enabled`;
    const retentionBucketName = `${state.bucketPrefix}-retention-after-creation`;

    try {
      await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
      await waitUntilBucketExists({ client }, { Bucket: noLockBucketName });
      await client.send(
        new CreateBucketCommand({
          Bucket: lockEnabledBucketName,
          ObjectLockEnabledForBucket: true,
        }),
      );
      await waitUntilBucketExists(
        { client },
        { Bucket: lockEnabledBucketName },
      );
      await client.send(
        new CreateBucketCommand({ Bucket: retentionBucketName }),
      );
      await waitUntilBucketExists({ client }, { Bucket: retentionBucketName });

      state.noLockBucketName = noLockBucketName;
      state.lockEnabledBucketName = lockEnabledBucketName;
      state.retentionBucketName = retentionBucketName;
    } catch (caught) {
      if (
        caught instanceof BucketAlreadyExists ||
        caught instanceof BucketAlreadyOwnedByYou
      ) {
        console.error(`${caught.name}: ${caught.message}`);
        state.earlyExit = true;
      } else {
        throw caught;
      }
    }
  })
};

```

```

    }
  }
});

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    (state) => `The following test files will be created:
      file0.txt in ${state.bucketPrefix}-no-lock.
      file1.txt in ${state.bucketPrefix}-no-lock.
      file0.txt in ${state.bucketPrefix}-lock-enabled.
      file1.txt in ${state.bucketPrefix}-lock-enabled.
      file0.txt in ${state.bucketPrefix}-retention-after-creation.
      file1.txt in ${state.bucketPrefix}-retention-after-creation.`
    , { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
    try {
      await client.send(
        new PutObjectCommand({
          Bucket: state.noLockBucketName,
          Key: "file0.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        })
      );
    }
  });

```

```
);
await client.send(
  new PutObjectCommand({
    Bucket: state.noLockBucketName,
    Key: "file1.txt",
    Body: "Content",
    ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
  }),
);
await client.send(
  new PutObjectCommand({
    Bucket: state.lockEnabledBucketName,
    Key: "file0.txt",
    Body: "Content",
    ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
  }),
);
await client.send(
  new PutObjectCommand({
    Bucket: state.lockEnabledBucketName,
    Key: "file1.txt",
    Body: "Content",
    ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
  }),
);
await client.send(
  new PutObjectCommand({
    Bucket: state.retentionBucketName,
    Key: "file0.txt",
    Body: "Content",
    ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
  }),
);
await client.send(
  new PutObjectCommand({
    Bucket: state.retentionBucketName,
    Key: "file1.txt",
    Body: "Content",
    ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
  }),
);
} catch (caught) {
  if (caught instanceof S3ServiceException) {
    console.error(
```

```

        `Error from S3 while uploading object.  ${caught.name}:
        ${caught.message}`,
        );
    } else {
        throw caught;
    }
}
});

/**
 * @param {Scenarios} scenarios
 */
const updateRetention = (scenarios) =>
    new scenarios.ScenarioOutput(
        "updateRetention",
        (state) => `A bucket can be configured to use object locking with a default
        retention period.
        A default retention period will be configured for ${state.bucketPrefix}-retention-
        after-creation.` ,
        { preformatted: true },
    );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateRetention = (scenarios) =>
    new scenarios.ScenarioInput(
        "confirmUpdateRetention",
        "Configure default retention period?",
        { type: "confirm" },
    );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateRetentionAction = (scenarios, client) =>
    new scenarios.ScenarioAction("updateRetentionAction", async (state) => {
        await client.send(
            new PutBucketVersioningCommand({
                Bucket: state.retentionBucketName,
                VersioningConfiguration: {
                    MFADelete: MFADeleteStatus.Disabled,
                    Status: BucketVersioningStatus.Enabled,

```

```

    },
  }),
);

const getBucketVersioning = new GetBucketVersioningCommand({
  Bucket: state.retentionBucketName,
});

await retry({ intervalInMs: 500, maxRetries: 10 }, async () => {
  const { Status } = await client.send(getBucketVersioning);
  if (Status !== "Enabled") {
    throw new Error("Bucket versioning is not enabled.");
  }
});

await client.send(
  new PutObjectLockConfigurationCommand({
    Bucket: state.retentionBucketName,
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
      Rule: {
        DefaultRetention: {
          Mode: "GOVERNANCE",
          Years: 1,
        },
      },
    },
  }),
);

/**
 * @param {Scenarios} scenarios
 */
const updateLockPolicy = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateLockPolicy",
    (state) => `Object lock policies can also be added to existing buckets.
An object lock policy will be added to ${state.bucketPrefix}-lock-enabled.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios

```

```

*/
const confirmUpdateLockPolicy = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateLockPolicy",
    "Add object lock policy?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateLockPolicyAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateLockPolicyAction", async (state) => {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.lockEnabledBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileEnabled",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
      ${state.lockEnabledBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileEnabledAction = (scenarios, client) =>

```

```

new scenarios.ScenarioAction(
  "setLegalHoldFileEnabledAction",
  async (state) => {
    await client.send(
      new PutObjectLegalHoldCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file0.txt",
        LegalHold: {
          Status: ObjectLockLegalHoldStatus.ON,
        },
      }),
    );
    console.log(
      `Modified legal hold for file0.txt in ${state.lockEnabledBucketName}.`,
    );
  },
  { skipWhen: (state) => !state.confirmSetLegalHoldFileEnabled },
);

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetRetentionPeriodFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileEnabled",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
      ${state.lockEnabledBucketName}?
      Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
      to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileEnabledAction",
    async (state) => {

```

```

    const retentionDate = new Date();
    retentionDate.setDate(retentionDate.getDate() + 1);
    await client.send(
      new PutObjectRetentionCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file1.txt",
        Retention: {
          Mode: ObjectLockRetentionMode.GOVERNANCE,
          RetainUntilDate: retentionDate,
        },
      }),
    );
    console.log(
      `Set retention for file1.txt in ${state.lockEnabledBucketName} until
    ${retentionDate.toISOString().split("T")[0]}.`,
    );
  },
  { skipWhen: (state) => !state.confirmSetRetentionPeriodFileEnabled },
);

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
    ${state.retentionBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileRetentionAction",
    async (state) => {
      await client.send(

```



```

        new PutObjectLegalHoldCommand({
            Bucket: state.retentionBucketName,
            Key: "file0.txt",
            LegalHold: {
                Status: ObjectLockLegalHoldStatus.ON,
            },
        }),
    );
    console.log(
        `Modified legal hold for file0.txt in ${state.retentionBucketName}.`,
    );
},
{ skipWhen: (state) => !state.confirmSetLegalHoldFileRetention },
);

/**
 * @param {Scenarios} scenarios
 */
const confirmSetRetentionPeriodFileRetention = (scenarios) =>
    new scenarios.ScenarioInput(
        "confirmSetRetentionPeriodFileRetention",
        (state) =>
            `Would you like to add a 1 day Governance retention period to file1.txt in
            ${state.retentionBucketName}?
            Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
            to delete this file or its bucket until the retention period has expired.`,
        {
            type: "confirm",
        },
    );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileRetentionAction = (scenarios, client) =>
    new scenarios.ScenarioAction(
        "setRetentionPeriodFileRetentionAction",
        async (state) => {
            const retentionDate = new Date();
            retentionDate.setDate(retentionDate.getDate() + 1);
            await client.send(
                new PutObjectRetentionCommand({
                    Bucket: state.retentionBucketName,

```

```

        Key: "file1.txt",
        Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
        },
        BypassGovernanceRetention: true,
    )),
    );
    console.log(
        `Set retention for file1.txt in ${state.retentionBucketName} until
        ${retentionDate.toISOString().split("T")[0]}.`,
    );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileRetention },
    );

export {
    getBucketPrefix,
    createBuckets,
    confirmCreateBuckets,
    createBucketsAction,
    populateBuckets,
    confirmPopulateBuckets,
    populateBucketsAction,
    updateRetention,
    confirmUpdateRetention,
    updateRetentionAction,
    updateLockPolicy,
    confirmUpdateLockPolicy,
    updateLockPolicyAction,
    confirmSetLegalHoldFileEnabled,
    setLegalHoldFileEnabledAction,
    confirmSetRetentionPeriodFileEnabled,
    setRetentionPeriodFileEnabledAction,
    confirmSetLegalHoldFileRetention,
    setLegalHoldFileRetentionAction,
    confirmSetRetentionPeriodFileRetention,
    setRetentionPeriodFileRetentionAction,
};

```

버킷의 파일을 조회하고 삭제합니다(repl.steps.js).

```

import {
  ChecksumAlgorithm,
  DeleteObjectCommand,
  GetObjectLegalHoldCommand,
  GetObjectLockConfigurationCommand,
  GetObjectRetentionCommand,
  ListObjectVersionsCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  DELETE_FILE: 2,
  DELETE_FILE_WITH_RETENTION: 3,
  OVERWRITE_FILE: 4,
  VIEW_RETENTION_SETTINGS: 5,
  VIEW_LEGAL_HOLD_SETTINGS: 6,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new scenarios.ScenarioInput(
    "replChoice",
    "Explore the S3 locking features by selecting one of the following choices",
    {
      type: "select",
      choices: [
        { name: "List all files in buckets", value: choices.LIST_ALL_FILES },
        { name: "Attempt to delete a file.", value: choices.DELETE_FILE },
        {
          name: "Attempt to delete a file with retention period bypass.",
          value: choices.DELETE_FILE_WITH_RETENTION,

```

```

    },
    { name: "Attempt to overwrite a file.", value: choices.OVERWRITE_FILE },
    {
      name: "View the object and bucket retention settings for a file.",
      value: choices.VIEW_RETENTION_SETTINGS,
    },
    {
      name: "View the legal hold settings for a file.",
      value: choices.VIEW_LEGAL_HOLD_SETTINGS,
    },
    { name: "Finish the workflow.", value: choices.EXIT },
  ],
},
);

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;
      files.push({ bucket, key: Key, version: VersionId });
    }
  }

  return files;
};

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const replAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [

```

```

    state.noLockBucketName,
    state.lockEnabledBucketName,
    state.retentionBucketName,
  ]);

  const fileInput = new scenarios.ScenarioInput(
    "selectedFile",
    "Select a file:",
    {
      type: "select",
      choices: files.map((file, index) => ({
        name: `${index + 1}: ${file.bucket}: ${file.key} (version: ${
          file.version
        })`,
        value: index,
      })),
    },
  );

  const { replChoice } = state;

  switch (replChoice) {
    case choices.LIST_ALL_FILES: {
      const files = await getAllFiles(client, [
        state.noLockBucketName,
        state.lockEnabledBucketName,
        state.retentionBucketName,
      ]);
      state.replOutput = files
        .map(
          (file) =>
            `${file.bucket}: ${file.key} (version: ${file.version})`,
        )
        .join("\n");
      break;
    }
    case choices.DELETE_FILE: {
      /** @type {number} */
      const fileToDelete = await fileInput.handle(state);
      const selectedFile = files[fileToDelete];
      try {
        await client.send(
          new DeleteObjectCommand({
            Bucket: selectedFile.bucket,

```

```

        Key: selectedFile.key,
        VersionId: selectedFile.version,
    )),
    );
    state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
    } catch (err) {
        state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
    }
    break;
}
case choices.DELETE_FILE_WITH_RETENTION: {
    /** @type {number} */
    const fileToDelete = await fileInput.handle(state);
    const selectedFile = files[fileToDelete];
    try {
        await client.send(
            new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
                BypassGovernanceRetention: true,
            )),
        );
        state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
    } catch (err) {
        state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
    }
    break;
}
case choices.OVERWRITE_FILE: {
    /** @type {number} */
    const fileToOverwrite = await fileInput.handle(state);
    const selectedFile = files[fileToOverwrite];
    try {
        await client.send(
            new PutObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                Body: "New content",
                ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
            )),
        );
        state.replOutput = `Overwrote ${selectedFile.key} in
${selectedFile.bucket} with new content.`;
    } catch (err) {
        state.replOutput = `Unable to overwrite object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
    }
    break;
}
}

```

```

        })),
    );
    state.replOutput = `Overwrote ${selectedFile.key} in
${selectedFile.bucket}.`;
    } catch (err) {
        state.replOutput = `Unable to overwrite object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
    }
    break;
}
case choices.VIEW_RETENTION_SETTINGS: {
    /** @type {number} */
    const fileToView = await fileInput.handle(state);
    const selectedFile = files[fileToView];
    try {
        const retention = await client.send(
            new GetObjectRetentionCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
            })),
    );
    const bucketConfig = await client.send(
        new GetObjectLockConfigurationCommand({
            Bucket: selectedFile.bucket,
        })),
    );
    state.replOutput = `Object retention for ${selectedFile.key}
in ${selectedFile.bucket}: ${retention.Retention?.Mode} until
${retention.Retention?.RetainUntilDate?.toISOString()}.
Bucket object lock config for ${selectedFile.bucket} in ${selectedFile.bucket}:
Enabled: ${bucketConfig.ObjectLockConfiguration?.ObjectLockEnabled}
Rule:
${JSON.stringify(bucketConfig.ObjectLockConfiguration?.Rule?.DefaultRetention)}`;
    } catch (err) {
        state.replOutput = `Unable to fetch object lock retention:
'${err.message}'`;
    }
    break;
}
case choices.VIEW_LEGAL_HOLD_SETTINGS: {
    /** @type {number} */
    const fileToView = await fileInput.handle(state);
    const selectedFile = files[fileToView];

```

```

    try {
      const legalHold = await client.send(
        new GetObjectLegalHoldCommand({
          Bucket: selectedFile.bucket,
          Key: selectedFile.key,
          VersionId: selectedFile.version,
        }),
      );
      state.replOutput = `Object legal hold for ${selectedFile.key} in
${selectedFile.bucket}: Status: ${legalHold.LegalHold?.Status}`;
    } catch (err) {
      state.replOutput = `Unable to fetch legal hold: '${err.message}'`;
    }
    break;
  }
  default:
    throw new Error(`Invalid replChoice: ${replChoice}`);
}
},
{
  whileConfig: {
    whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
    input: replInput(scenarios),
    output: new scenarios.ScenarioOutput(
      "REPL output",
      (state) => state.replOutput,
      { preformatted: true },
    ),
  },
},
);

export { replInput, replAction, choices };

```

생성된 모든 리소스를 삭제합니다(clean.steps.js).

```

import {
  DeleteObjectCommand,
  DeleteBucketCommand,
  ListObjectVersionsCommand,
  GetObjectLegalHoldCommand,
  GetObjectRetentionCommand,

```



```

    PutObjectLegalHoldCommand,
  } from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { noLockBucketName, lockEnabledBucketName, retentionBucketName } =
      state;

    const buckets = [
      noLockBucketName,
      lockEnabledBucketName,
      retentionBucketName,
    ];

    for (const bucket of buckets) {
      /** @type {import("@aws-sdk/client-s3").ListObjectVersionsCommandOutput} */
      let objectsResponse;

      try {
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
        );
      }
    }
  });

```

```
    } catch (e) {
      if (e instanceof Error && e.name === "NoSuchBucket") {
        console.log("Object's bucket has already been deleted.");
        continue;
      }
      throw e;
    }

    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;

      try {
        const legalHold = await client.send(
          new GetObjectLegalHoldCommand({
            Bucket: bucket,
            Key,
            VersionId,
          }),
        );

        if (legalHold.LegalHold?.Status === "ON") {
          await client.send(
            new PutObjectLegalHoldCommand({
              Bucket: bucket,
              Key,
              VersionId,
              LegalHold: {
                Status: "OFF",
              },
            }),
          );
        }
      } catch (err) {
        console.log(
          `Unable to fetch legal hold for ${Key} in ${bucket}: '${err.message}'`,
        );
      }

      try {
        const retention = await client.send(
          new GetObjectRetentionCommand({
            Bucket: bucket,
            Key,
            VersionId,
```

```

    }},
  );

  if (retention.Retention?.Mode === "GOVERNANCE") {
    await client.send(
      new DeleteObjectCommand({
        Bucket: bucket,
        Key,
        VersionId,
        BypassGovernanceRetention: true,
      }),
    );
  }
} catch (err) {
  console.log(
    `Unable to fetch object lock retention for ${Key} in ${bucket}:`
    `${err.message}`
  );
}

await client.send(
  new DeleteObjectCommand({
    Bucket: bucket,
    Key,
    VersionId,
  }),
);
}

await client.send(new DeleteBucketCommand({ Bucket: bucket }));
console.log(`Delete for ${bucket} complete.`);
}
});

export { confirmCleanup, cleanupAction };

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [GetObjectLegalHold](#)
 - [GetObjectLockConfiguration](#)
 - [GetObjectRetention](#)
 - [PutObjectLegalHold](#)

- [PutObjectLockConfiguration](#)
- [PutObjectRetention](#)

조건부 요청 수행

다음 코드 예제에서는 Amazon S3 요청에 사전 조건을 추가하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

워크폴로의 진입점입니다(index.js). 모든 단계를 오케스트레이션합니다. GitHub를 방문하여 Scenario, ScenarioInput, ScenarioOutput, ScenarioAction에 대한 구현 세부 정보를 확인하세요.

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "../welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
} from "../setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
```

```
const client = new S3Client({});

return {
  deploy: new scenarios.Scenario(
    "S3 Conditional Requests - Deploy",
    [
      welcome(scenarios),
      welcomeContinue(scenarios),
      exitOnFalse(scenarios, "welcomeContinue"),
      getBucketPrefix(scenarios),
      createBuckets(scenarios),
      confirmCreateBuckets(scenarios),
      exitOnFalse(scenarios, "confirmCreateBuckets"),
      createBucketsAction(scenarios, client),
      populateBuckets(scenarios),
      confirmPopulateBuckets(scenarios),
      exitOnFalse(scenarios, "confirmPopulateBuckets"),
      populateBucketsAction(scenarios, client),
      saveState,
    ],
    initialState,
  ),
  demo: new scenarios.Scenario(
    "S3 Conditional Requests - Demo",
    [loadState, welcome(scenarios), replAction(scenarios, client)],
    initialState,
  ),
  clean: new scenarios.Scenario(
    "S3 Conditional Requests - Destroy",
    [
      loadState,
      confirmCleanup(scenarios),
      exitOnFalse(scenarios, "confirmCleanup"),
      cleanupAction(scenarios, client),
    ],
    initialState,
  ),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "../clean.steps.js";
```

```
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios, {
    name: "Amazon S3 object locking workflow",
    description:
      "Work with Amazon Simple Storage Service (Amazon S3) object locking features.",
    synopsis:
      "node index.js --scenario <deploy | demo | clean> [-h|--help] [-y|--yes] [-v|--verbose]",
  });
}
```

환영 메시지를 콘솔에 출력합니다(welcome.steps.js).

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    "This example demonstrates the use of conditional requests for S3 operations." +
    " You can use conditional requests to add preconditions to S3 read requests to return " +
    "or copy an object based on its Entity tag (ETag), or last modified date.You can use " +
    "a conditional write requests to prevent overwrites by ensuring there is no existing " +
    "object with the same key.\n" +
    "This example will enable you to perform conditional reads and writes that will succeed " +
    "or fail based on your selected options.\n" +
    "Sample buckets and a sample object will be created as part of the example.\n" +
    "Some steps require a key name prefix to be defined by the user. Before you begin, you can " +
```

```

    "optionally edit this prefix in ./object_name.json. If you do so, please
    reload the scenario before you begin.",
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };

```

버킷 및 객체를 배포합니다(setup.steps.js).

```

import {
  ChecksumAlgorithm,
  CreateBucketCommand,
  PutObjectCommand,
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  S3ServiceException,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const getBucketPrefix = (scenarios) =>
  new scenarios.ScenarioInput(

```

```

    "bucketPrefix",
    "Provide a prefix that will be used for bucket creation.",
    { type: "input", default: "amzn-s3-demo-bucket" },
  );
/**
 * @param {Scenarios} scenarios
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    (state) => `The following buckets will be created:
      ${state.bucketPrefix}-source-bucket.
      ${state.bucketPrefix}-destination-bucket.` ,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const sourceBucketName = `${state.bucketPrefix}-source-bucket`;
    const destinationBucketName = `${state.bucketPrefix}-destination-bucket`;

    try {
      await client.send(
        new CreateBucketCommand({
          Bucket: sourceBucketName,
        }),
      );
      await waitUntilBucketExists({ client }, { Bucket: sourceBucketName });
      await client.send(
        new CreateBucketCommand({
          Bucket: destinationBucketName,
        }),
      );
    }
  });

```



```

    );
    await waitUntilBucketExists(
      { client },
      { Bucket: destinationBucketName },
    );

    state.sourceBucketName = sourceBucketName;
    state.destinationBucketName = destinationBucketName;
  } catch (caught) {
    if (
      caught instanceof BucketAlreadyExists ||
      caught instanceof BucketAlreadyOwnedByYou
    ) {
      console.error(`${caught.name}: ${caught.message}`);
      state.earlyExit = true;
    } else {
      throw caught;
    }
  }
});

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    (state) => `The following test files will be created:
      file01.txt in ${state.bucketPrefix}-source-bucket.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios

```

```

* @param {S3Client} client
*/
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
    try {
      await client.send(
        new PutObjectCommand({
          Bucket: state.sourceBucketName,
          Key: "file01.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
    } catch (caught) {
      if (caught instanceof S3ServiceException) {
        console.error(
          `Error from S3 while uploading object.  ${caught.name}:
${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  });

export {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
};

```

S3 조건부 요청을 사용하여 객체를 가져오고, 복사하고, 업로드합니다(repl.steps.js).

```

import path from "node:path";
import { fileURLToPath } from "node:url";
import { dirname } from "node:path";

```

```

import {
  ListObjectVersionsCommand,
  GetObjectCommand,
  CopyObjectCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";
import data from "./object_name.json" assert { type: "json" };
import { readFile } from "node:fs/promises";
import {
  ScenarioInput,
  Scenario,
  ScenarioAction,
  ScenarioOutput,
} from "../../libs/scenario/index.js";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  CONDITIONAL_READ: 2,
  CONDITIONAL_COPY: 3,
  CONDITIONAL_WRITE: 4,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new ScenarioInput(
    "replChoice",
    "Explore the S3 conditional request features by selecting one of the following choices",
    {
      type: "select",
      choices: [
        { name: "Print list of bucket items.", value: choices.LIST_ALL_FILES },
        {

```

```

        name: "Perform a conditional read.",
        value: choices.CONDITIONAL_READ,
      },
      {
        name: "Perform a conditional copy. These examples use the key name prefix
defined in ./object_name.json.",
        value: choices.CONDITIONAL_COPY,
      },
      {
        name: "Perform a conditional write. This example use the sample file ./
text02.txt.",
        value: choices.CONDITIONAL_WRITE,
      },
      { name: "Finish the workflow.", value: choices.EXIT },
    ],
  },
);

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket })),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key } = version;
      files.push({ bucket, key: Key });
    }
  }
  return files;
};

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 * @param {string} key
 */
const getEtag = async (client, bucket, key) => {
  const objectsResponse = await client.send(

```

```

    new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    }),
  );
  return objectsResponse.ETag;
};

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
export const replAction = (scenarios, client) =>
  new ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.sourceBucketName,
        state.destinationBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file to use:",
        {
          type: "select",
          choices: files.map((file, index) => ({
            name: `${index + 1}: ${file.bucket}: ${file.key} (Etag: ${
              file.version
            })`,
            value: index,
          })),
        },
      );

      const condReadOptions = new scenarios.ScenarioInput(
        "selectOption",
        "Which conditional read action would you like to take?",
        {
          type: "select",

```

```

        choices: [
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",
            "If-Modified-Since: using yesterday's date. This condition should
succeed.",
            "If-Unmodified-Since: using yesterday's date. This condition should
fail.",
        ],
    },
);
const condCopyOptions = new scenarios.ScenarioInput(
    "selectOption",
    "Which conditional copy action would you like to take?",
    {
        type: "select",
        choices: [
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",
            "If-Modified-Since: using yesterday's date. This condition should
succeed.",
            "If-Unmodified-Since: using yesterday's date. This condition should
fail.",
        ],
    },
);
const condWriteOptions = new scenarios.ScenarioInput(
    "selectOption",
    "Which conditional write action would you like to take?",
    {
        type: "select",
        choices: [
            "IfNoneMatch condition on the object key: If the key is a duplicate, the
write will fail.",
        ],
    },
);

const { replChoice } = state;

switch (replChoice) {
    case choices.LIST_ALL_FILES: {
        const files = await getAllFiles(client, [
            state.sourceBucketName,
            state.destinationBucketName,

```

```
]);
state.replOutput = files
  .map(
    (file) => `Items in bucket ${file.bucket}: object: ${file.key} `,
  )
  .join("\n");
break;
}
case choices.CONDITIONAL_READ:
{
  const selectedCondRead = await condReadOptions.handle(state);
  if (
    selectedCondRead ===
    "If-Match: using the object's ETag. This condition should succeed."
  ) {
    const bucket = state.sourceBucketName;
    const key = "file01.txt";
    const ETag = await getEtag(client, bucket, key);

    try {
      await client.send(
        new GetObjectCommand({
          Bucket: bucket,
          Key: key,
          IfMatch: ETag,
        }),
      );
      state.replOutput = `${key} in bucket ${state.sourceBucketName} read
because ETag provided matches the object's ETag.`;
    } catch (err) {
      state.replOutput = `Unable to read object ${key} in bucket
${state.sourceBucketName}: ${err.message}`;
    }
    break;
  }
  if (
    selectedCondRead ===
    "If-None-Match: using the object's ETag. This condition should fail."
  ) {
    const bucket = state.sourceBucketName;
    const key = "file01.txt";
    const ETag = await getEtag(client, bucket, key);

    try {
```

```

        await client.send(
            new GetObjectCommand({
                Bucket: bucket,
                Key: key,
                IfNoneMatch: ETag,
            }),
        );
        state.replOutput = `${key} in ${state.sourceBucketName} was
returned.`;
    } catch (err) {
        state.replOutput = `${key} in ${state.sourceBucketName} was not
read: ${err.message}`;
    }
    break;
}
if (
    selectedCondRead ===
    "If-Modified-Since: using yesterday's date. This condition should
succeed."
) {
    const date = new Date();
    date.setDate(date.getDate() - 1);

    const bucket = state.sourceBucketName;
    const key = "file01.txt";
    try {
        await client.send(
            new GetObjectCommand({
                Bucket: bucket,
                Key: key,
                IfModifiedSince: date,
            }),
        );
        state.replOutput = `${key} in bucket ${state.sourceBucketName} read
because it has been created or modified in the last 24 hours.`;
    } catch (err) {
        state.replOutput = `Unable to read object ${key} in bucket
${state.sourceBucketName}: ${err.message}`;
    }
    break;
}
if (
    selectedCondRead ===

```



```

        "If-Unmodified-Since: using yesterday's date. This condition should
fail."
    ) {
        const bucket = state.sourceBucketName;
        const key = "file01.txt";

        const date = new Date();
        date.setDate(date.getDate() - 1);
        try {
            await client.send(
                new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfUnmodifiedSince: date,
                }),
            );
            state replOutput = `${key} in ${state.sourceBucketName} was read.`;
        } catch (err) {
            state replOutput = `${key} in ${state.sourceBucketName} was not
read: ${err.message}`;
        }
        break;
    }
    break;
case choices.CONDITIONAL_COPY: {
    const selectedCondCopy = await condCopyOptions.handle(state);
    if (
        selectedCondCopy ===
        "If-Match: using the object's ETag. This condition should succeed."
    ) {
        const bucket = state.sourceBucketName;
        const key = "file01.txt";
        const ETag = await getEtag(client, bucket, key);

        const copySource = `${bucket}/${key}`;
        // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
        const name = data.name;
        const copiedKey = `${name}${key}`;
        try {
            await client.send(
                new CopyObjectCommand({
                    CopySource: copySource,

```

```

        Bucket: state.destinationBucketName,
        Key: copiedKey,
        CopySourceIfMatch: ETag,
    )),
    );
    state.replOutput = `${key} copied as ${copiedKey} to bucket
${state.destinationBucketName} because ETag provided matches the object's ETag.`;
    } catch (err) {
        state.replOutput = `Unable to copy object ${key} as ${copiedKey} to
bucket ${state.destinationBucketName}: ${err.message}`;
    }
    break;
}
if (
    selectedCondCopy ===
    "If-None-Match: using the object's ETag. This condition should fail."
) {
    const bucket = state.sourceBucketName;
    const key = "file01.txt";
    const ETag = await getEtag(client, bucket, key);
    const copySource = `${bucket}/${key}`;
    // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
    const name = data.name;
    const copiedKey = `${name}${key}`;

    try {
        await client.send(
            new CopyObjectCommand({
                CopySource: copySource,
                Bucket: state.destinationBucketName,
                Key: copiedKey,
                CopySourceIfNoneMatch: ETag,
            )),
        );
        state.replOutput = `${copiedKey} copied to bucket
${state.destinationBucketName}`;
    } catch (err) {
        state.replOutput = `Unable to copy object as ${key} as as ${copiedKey}
to bucket ${state.destinationBucketName}: ${err.message}`;
    }
    break;
}
if (

```

```

        selectedCondCopy ===
        "If-Modified-Since: using yesterday's date. This condition should
succeed."
    ) {
        const bucket = state.sourceBucketName;
        const key = "file01.txt";
        const copySource = `${bucket}/${key}`;
        // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
        const name = data.name;
        const copiedKey = `${name}${key}`;

        const date = new Date();
        date.setDate(date.getDate() - 1);

        try {
            await client.send(
                new CopyObjectCommand({
                    CopySource: copySource,
                    Bucket: state.destinationBucketName,
                    Key: copiedKey,
                    CopySourceIfModifiedSince: date,
                }),
            );
            state.replOutput = `${key} copied as ${copiedKey} to bucket
${state.destinationBucketName} because it has been created or modified in the last
24 hours.`;
        } catch (err) {
            state.replOutput = `Unable to copy object ${key} as ${copiedKey} to
bucket ${state.destinationBucketName} : ${err.message}`;
        }
        break;
    }
    if (
        selectedCondCopy ===
        "If-Unmodified-Since: using yesterday's date. This condition should
fail."
    ) {
        const bucket = state.sourceBucketName;
        const key = "file01.txt";
        const copySource = `${bucket}/${key}`;
        // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
        const name = data.name;

```

```

    const copiedKey = `${name}${key}`;

    const date = new Date();
    date.setDate(date.getDate() - 1);

    try {
      await client.send(
        new CopyObjectCommand({
          CopySource: copySource,
          Bucket: state.destinationBucketName,
          Key: copiedKey,
          CopySourceIfUnmodifiedSince: date,
        }),
      );
      state.replOutput = `${copiedKey} copied to bucket
${state.destinationBucketName} because it has not been created or modified in the
last 24 hours.`;
    } catch (err) {
      state.replOutput = `Unable to copy object ${key} to bucket
${state.destinationBucketName}: ${err.message}`;
    }
    break;
  }
  case choices.CONDITIONAL_WRITE:
    {
      const selectedCondWrite = await condWriteOptions.handle(state);
      if (
        selectedCondWrite ===
        "IfNoneMatch condition on the object key: If the key is a duplicate,
the write will fail."
      ) {
        // Optionally edit the default key name prefix of the copied object
        in ./object_name.json.
        const key = "text02.txt";
        const __filename = fileURLToPath(import.meta.url);
        const __dirname = dirname(__filename);
        const filePath = path.join(__dirname, "text02.txt");
        try {
          await client.send(
            new PutObjectCommand({
              Bucket: `${state.destinationBucketName}`,
              Key: `${key}`,
              Body: await readFile(filePath),

```

```

                IfNoneMatch: "*",
            })),
        );
        state.replOutput = `${key} uploaded to bucket
${state.destinationBucketName} because the key is not a duplicate.`;
    } catch (err) {
        state.replOutput = `Unable to upload object to bucket
${state.destinationBucketName}:${err.message}`;
    }
    break;
}
}
break;

default:
    throw new Error(`Invalid replChoice: ${replChoice}`);
}
},
{
    whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new ScenarioOutput("REPL output", (state) => state.replOutput, {
            preformatted: true,
        }),
    },
},
);

export { replInput, choices };

```

생성된 모든 리소스를 삭제합니다(clean.steps.js).

```

import {
    DeleteObjectCommand,
    DeleteBucketCommand,
    ListObjectVersionsCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

```

```
/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { sourceBucketName, destinationBucketName } = state;
    const buckets = [sourceBucketName, destinationBucketName].filter((b) => b);

    for (const bucket of buckets) {
      try {
        let objectsResponse;
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
        );
        for (const version of objectsResponse.Versions || []) {
          const { Key, VersionId } = version;
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: bucket,
                Key,
                VersionId,
              }),
            );
          } catch (err) {
            console.log(`An error occurred: ${err.message} `);
          }
        }
      }
    }
  })
```

```

    } catch (e) {
      if (e instanceof Error && e.name === "NoSuchBucket") {
        console.log("Objects and buckets have already been deleted.");
        continue;
      }
      throw e;
    }

    await client.send(new DeleteBucketCommand({ Bucket: bucket }));
    console.log(`Delete for ${bucket} complete.`);
  }
});

export { confirmCleanup, cleanupAction };

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 항목을 참조하세요.

- [CopyObject](#)
- [GetObject](#)
- [PutObject](#)

대용량 파일 업로드 또는 다운로드

다음 코드 예제는 Amazon S3에 대용량 파일을 업로드하고 Amazon S3에서 대용량 파일을 다운로드 하는 방법을 보여줍니다.

자세한 내용은 [멀티파트 업로드를 사용하여 객체 업로드](#)를 참조하세요.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

대용량 파일을 업로드합니다.

```

import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

```

```
import {
  ProgressBar,
  logger,
} from "@aws-doc-sdk-examples/lib/utils/util-log.js";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

/**
 * Create a 25MB file and upload it in parts to the specified
 * Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const str = createString();
  const buffer = Buffer.from(str, "utf8");
  const progressBar = new ProgressBar({
    description: `Uploading "${key}" to "${bucketName}"`,
    barLength: 30,
  });

  try {
    const upload = new Upload({
      client: new S3Client({}),
      params: {
        Bucket: bucketName,
        Key: key,
        Body: buffer,
      },
    });

    upload.on("httpUploadProgress", ({ loaded, total }) => {
      progressBar.update({ current: loaded, total });
    });

    await upload.done();
  } catch (caught) {
    if (caught instanceof Error && caught.name === "AbortError") {
      logger.error(`Multipart upload was aborted. ${caught.message}`);
    } else {
      throw caught;
    }
  }
}
```



```

    }
  }
};

```

대용량 파일을 다운로드합니다.

```

import { fileURLToPath } from "node:url";
import { GetObjectCommand, NoSuchKey, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream, rmSync } from "node: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);
};

/**
 * @param {string | undefined} contentRange
 */
export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: Number.parseInt(start),
    end: Number.parseInt(end),
    length: Number.parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

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 };

  const { ContentRange, Body } = await getObjectRange({
    bucket,
    key,
    ...nextRange,
  });
  console.log(`Downloaded bytes ${nextRange.start} to ${nextRange.end}`);

  writeStream.write(await Body.transformToByteArray());
  rangeAndLength = getRangeAndLength(ContentRange);
}
};

/**
 * Download a large object from and Amazon S3 bucket.
 *
 * 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.
 *
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  try {
    await downloadInChunks({
      bucket: bucketName,
      key: key,
    });
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(`Failed to download object. No such key "${key}".`);
      rmSync(key);
    }
  }
};

```

서버리스 예제

Amazon S3 트리거를 사용하여 Lambda 함수 간접 호출

다음 코드 예제는 S3 버킷에 객체를 업로드하여 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 해당 함수는 이벤트 파라미터에서 S3 버킷 이름과 객체 키를 검색하고 Amazon S3 API를 호출하여 객체의 콘텐츠 유형을 검색하고 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 S3 이벤트를 사용합니다.

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

const client = new S3Client();

export const handler = async (event, context) => {

  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));

  try {
    const { ContentType } = await client.send(new HeadObjectCommand({
      Bucket: bucket,
      Key: key,
    }));

    console.log('CONTENT TYPE:', ContentType);
    return ContentType;

  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure they exist and your bucket is in the same region as this function.`;
  }
}
```

```

        console.log(message);
        throw new Error(message);
    }
};

```

TypeScript를 사용하여 Lambda로 S3 이벤트를 사용합니다.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
    const params = {
        Bucket: bucket,
        Key: key,
    };
    try {
        const { ContentType } = await s3.send(new HeadObjectCommand(params));
        console.log('CONTENT TYPE:', ContentType);
        return ContentType;
    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}. Make sure they exist and your bucket is in the same region as this function.`;
        console.log(message);
        throw new Error(message);
    }
};

```

SDK for JavaScript(v3)를 사용한 SageMaker AI 예제

다음 코드 예제에서는 SageMaker AI와 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)
- [시나리오](#)

시작하기

SageMaker AI 시작

다음 코드 예제에서는 SageMaker AI 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 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;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [ListNotebookInstances](#)를 참조하세요.

작업

CreatePipeline

다음 코드 예시는 CreatePipeline의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

로컬에서 제공된 JSON 정의를 사용하여 SageMaker AI 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,
    )}../../../../scenarios/features/sagemaker_pipelines/resources/
    GeoSpatialPipeline.json`,
  )
  .toString()
  .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
```

```
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
    })),
    );

    try {
        const { PipelineArn } = await createPipeline();
        arn = PipelineArn;
    } catch (caught) {
        if (
            caught instanceof Error &&
            caught.name === "ValidationException" &&
            caught.message.includes(
                "Pipeline names must be unique within an AWS account and region",
            )
        ) {
            const { PipelineArn } = await sagemakerClient.send(
                new DescribePipelineCommand({ PipelineName: name }),
            );
            arn = PipelineArn;
        } else {
            throw caught;
        }
    }

    return {
        arn,
        cleanUp: async () => {
            await sagemakerClient.send(
                new DeletePipelineCommand({ PipelineName: name }),
            );
        },
    };
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreatePipeline](#)을 참조하세요.

DeletePipeline

다음 코드 예시는 DeletePipeline의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

SageMaker AI 파이프라인을 삭제하기 위한 구문입니다. 이 코드는 더 큰 함수의 일부입니다. 자세한 내용은 '파이프라인 생성' 또는 GitHub 리포지토리를 참조하세요.

```
await sagemakerClient.send(
  new DeletePipelineCommand({ PipelineName: name }),
);
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeletePipeline](#)을 참조하세요.

DescribePipelineExecution

다음 코드 예시는 DescribePipelineExecution의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

SageMaker AI 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, wait: (ms: number) => Promise<void> }} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
```

```
const command = new DescribePipelineExecutionCommand({
  PipelineExecutionArn: arn,
});

let complete = false;
const 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);
}
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribePipelineExecution](#)을 참조하세요.

StartPipelineExecution

다음 코드 예시는 StartPipelineExecution의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

SageMaker AI 파이프라인 실행을 시작합니다.

```
/**
 * 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,
};
}
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [StartPipelineExecution](#)을 참조하세요.

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. AWS SDKs를 사용하여 SageMaker 파이프라인 생성 및 실행을 참조하세요](#).

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

다음 파일 발췌문에는 SageMaker AI 클라이언트를 사용하여 파이프라인을 관리하는 함수가 포함되어 있습니다.

```
import { readFileSync } from "node:fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
  GetRoleCommand,
  ListPoliciesCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
  GetFunctionCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
```

```

} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
  GetQueueUrlCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry } 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 createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: { Service: ["lambda.amazonaws.com"] },
            },
          ],
        }),
      }),
    );

  let role = null;

  try {
    const { Role } = await createRole();
    role = Role;
  } catch (caught) {

```

```

    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }
}

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 policyConfig = {
    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 createPolicy = () =>
    iamClient.send(
        new CreatePolicyCommand({
            PolicyDocument: JSON.stringify(policyConfig),
            PolicyName: name,
        }),
    );

let policy = null;

try {

```

```

    const { Policy } = await createPolicy();
    policy = Policy;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    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`;

  // If a function of the same name already exists, return that
  // function's ARN instead. By default this is
  // "sagemaker-wkflw-lambda-function", so collisions are
  // unlikely.
  const createFunction = async () => {
    try {
      return await lambdaClient.send(
        new CreateFunctionCommand({
          Code: {
            ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
          },
          Runtime: Runtime.nodejs18x,
          Handler: "index.handler",
          Layers: [layerVersionArn],
          FunctionName: name,
          Role: roleArn,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "ResourceConflictException"
      ) {
        const { Configuration } = await lambdaClient.send(
          new GetFunctionCommand({ FunctionName: name }),
        );
        return Configuration;
      }
      throw caught;
    }
  }
}

```

```

};

// 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 },
  createFunction,
);

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,
  )}../../../../scenarios/features/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, wait:
(ms: number) => Promise<void>}} props
*/
export async function createSagemakerRole({ name, iamClient, wait }) {
  let role = null;

  const createRole = () =>
    iamClient.send(
      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",
                ],
              },
            },
          ],
        })),
    );

  try {
    const { Role } = await createRole();
    role = Role;
    // Wait for the role to be ready.
    await wait(10);
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }
}

```

```

    }
  }

  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 policyConfig = {
    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 createPolicy = () =>
  iamClient.send(
    new CreatePolicyCommand({
      PolicyDocument: JSON.stringify(policyConfig),
      PolicyName: name,
    }),
  );

let policy = null;

try {
  const { Policy } = await createPolicy();
  policy = Policy;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
    if (Policies) {
      policy = Policies.find((p) => p.PolicyName === name);
    } else {
      throw new Error("No policies found.");
    }
  } else {
    throw caught;
  }
}

return {
  arn: policy?.Arn,
  policyConfig,
  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,
    )}../../../../../scenarios/features/sagemaker_pipelines/resources/
    GeoSpatialPipeline.json`,
    )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&

```

```

        caught.name === "ValidationException" &&
        caught.message.includes(
            "Pipeline names must be unique within an AWS account and region",
        )
    ) {
        const { PipelineArn } = await sagemakerClient.send(
            new DescribePipelineCommand({ PipelineName: name }),
        );
        arn = PipelineArn;
    } else {
        throw caught;
    }
}

return {
    arn,
    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 createSqsQueue = () =>
        sqsClient.send(
            new CreateQueueCommand({
                QueueName: name,
                Attributes: {
                    DelaySeconds: "5",
                    ReceiveMessageWaitTimeSeconds: "5",
                    VisibilityTimeout: "300",
                },
            }),
        );

    let queueUrl = null;
    try {

```

```

    const { QueueUrl } = await createSqsQueue();
    queueUrl = QueueUrl;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "QueueNameExists") {
      const { QueueUrl } = await sqsClient.send(
        new GetQueueUrlCommand({ QueueName: name }),
      );
      queueUrl = QueueUrl;
    } else {
      throw caught;
    }
  }
}

const { Attributes } = await retry(
  { intervalInMs: 1000, maxRetries: 60 },
  () =>
    sqsClient.send(
      new GetQueueAttributesCommand({
        QueueUrl: queueUrl,
        AttributeNames: ["QueueArn"],
      }),
    ),
);

return {
  queueUrl,
  queueArn: Attributes.QueueArn,
  cleanUp: async () => {
    await sqsClient.send(new DeleteQueueCommand({ QueueUrl: queueUrl }));
  },
};
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{
 *   paginateListEventSourceMappings: () => Generator<import('@aws-sdk/client-
lambda').ListEventSourceMappingsCommandOutput>,
 *   lambdaName: string,
 *   queueArn: string,
 *   lambdaClient: import('@aws-sdk/client-lambda').LambdaClient}} props
 */
export async function configureLambdaSQSEventSource({

```

```

    lambdaName,
    queueArn,
    lambdaClient,
    paginateListEventSourceMappings,
  }) {
    let uuid = null;
    const createEvenSourceMapping = () =>
      lambdaClient.send(
        new CreateEventSourceMappingCommand({
          EventSourceArn: queueArn,
          FunctionName: lambdaName,
        }),
      );

    try {
      const { UUID } = await createEvenSourceMapping();
      uuid = UUID;
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "ResourceConflictException"
      ) {
        const paginator = paginateListEventSourceMappings(
          { client: lambdaClient },
          {},
        );
        /**
         * @type {import('@aws-sdk/client-lambda').EventSourceMappingConfiguration[]}
         */
        const eventSourceMappings = [];
        for await (const page of paginator) {
          eventSourceMappings.concat(page.EventSourceMappings || []);
        }

        const { Configuration } = await lambdaClient.send(
          new GetFunctionCommand({ FunctionName: lambdaName }),
        );

        uuid = eventSourceMappings.find(
          (mapping) =>
            mapping.EventSourceArn === queueArn &&
            mapping.FunctionArn === Configuration.FunctionArn,
        ).UUID;
      } else {

```

```

        throw caught;
    }
}

return {
  cleanUp: async () => {
    await lambdaClient.send(
      new DeleteEventSourceMappingCommand({
        UUID: 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,
 *   paginateListObjectsV2: () => Generator<import('@aws-sdk/client-
s3').ListObjectsCommandOutput>
 * }} props
 */
export async function createS3Bucket({
  name,
  s3Client,
  paginateListObjectsV2,
}) {
  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, wait: (ms: number) => Promise<void> }} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  const 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 앞의 라이브러리 함수를 사용하여 SageMaker AI 파이프라인을 설정하고 실행하며 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 {
      this.logger.logSeparator();
      const doCleanUp = await this.prompter.confirm({
        message: "Clean up resources?",
      });
      if (doCleanUp) {
        await this.cleanUp();
      }
    }
  }

  async cleanUp() {
    // Run all of the clean up functions. If any fail, we log the error and
    continue.
    // This ensures all clean up functions are run.
    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,
    wait,
```

```
});  
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,
        wait,
    });

    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 항목을 참조하세요.
 - [CreatePipeline](#)
 - [DeletePipeline](#)
 - [DescribePipelineExecution](#)
 - [StartPipelineExecution](#)
 - [UpdatePipeline](#)

SDK for JavaScript (v3)를 사용한 Secrets Manager 예

다음 코드 예제에서는 Secrets Manager에서 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

GetSecretValue

다음 코드 예시는 GetSecretValue의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 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 참조의 [GetSecretValue](#)를 참조하세요.

SDK for JavaScript (v3)를 사용한 Amazon SES 예

다음 코드 예제에서는 Amazon SES에서 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)
- [시나리오](#)

작업

CreateReceiptFilter

다음 코드 예시는 CreateReceiptFilter의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateReceiptFilter](#)를 참조하세요.

CreateReceiptRule

다음 코드 예시는 CreateReceiptRule의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateReceiptRule](#)을 참조하세요.

CreateReceiptRuleSet

다음 코드 예시는 CreateReceiptRuleSet의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateReceiptRuleSet](#)를 참조하세요.

CreateTemplate

다음 코드 예시는 CreateTemplate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateTemplate](#)을 참조하세요.

DeleteIdentity

다음 코드 예시는 DeleteIdentity의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteIdentity](#)를 참조하세요.

DeleteReceiptFilter

다음 코드 예시는 DeleteReceiptFilter의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteReceiptFilter](#)를 참조하세요.

DeleteReceiptRule

다음 코드 예시는 DeleteReceiptRule의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteReceiptRule](#)을 참조하세요.

DeleteReceiptRuleSet

다음 코드 예시는 DeleteReceiptRuleSet의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utlils/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 참조의 [DeleteReceiptRuleSet](#)를 참조하세요.

DeleteTemplate

다음 코드 예시는 DeleteTemplate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteTemplate](#)을 참조하세요.

GetTemplate

다음 코드 예시는 GetTemplate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetTemplate](#)을 참조하세요.

ListIdentities

다음 코드 예시는 ListIdentities의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListIdentities](#)를 참조하세요.

ListReceiptFilters

다음 코드 예시는 ListReceiptFilters의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListReceiptFilters](#)를 참조하세요.

ListTemplates

다음 코드 예시는 ListTemplates의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListTemplates](#)를 참조하세요.

SendBulkTemplatedEmail

다음 코드 예시는 SendBulkTemplatedEmail의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
}

```

```
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [SendBulkTemplatedEmail](#)을 참조하세요.

SendEmail

다음 코드 예시는 SendEmail의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 (caught) {
        if (caught instanceof Error && caught.name === "MessageRejected") {
            /** @type { import('@aws-sdk/client-ses').MessageRejected } */
            const messageRejectedError = caught;
            return messageRejectedError;
        }
        throw caught;
    }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [SendEmail](#)을 참조하세요.

SendRawEmail

다음 코드 예시는 SendRawEmail의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

[nodemailer](#)를 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 참조의 [SendRawEmail](#)을 참조하세요.

SendTemplatedEmail

다음 코드 예시는 SendTemplatedEmail의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [SendTemplatedEmail](#)을 참조하세요.

UpdateTemplate

다음 코드 예시는 UpdateTemplate의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [UpdateTemplate](#)을 참조하세요.

VerifyDomainIdentity

다음 코드 예시는 VerifyDomainIdentity의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [VerifyDomainIdentity](#)를 참조하세요.

VerifyEmailIdentity

다음 코드 예시는 VerifyEmailIdentity의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 SDK API 참조의 [VerifyEmailIdentity](#)를 참조하세요.

시나리오

Amazon Transcribe 스트리밍 앱 구축

다음 코드 예제에서는 라이브 오디오를 실시간으로 기록, 변환 및 번역하고 결과를 이메일로 보내는 앱을 구축하는 방법을 보여줍니다.

SDK for JavaScript (v3)

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 [GitHub](#)에서 전체 예시를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Aurora 서버리스 작업 항목 트래커 만들기

다음 코드 예제에서는 Amazon Aurora Serverless 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 사용하여 보고서를 보내는 웹 애플리케이션을 만드는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 보여줍니다. 이 예제에서는 Express Node.js 백엔드와의 상호 작용을 위해 React.js로 빌드된 프론트엔드를 사용합니다.

- React.js 웹 애플리케이션을와 통합합니다 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

이미지에서 객체 감지

다음 코드 예제에서는 Amazon Rekognition을 사용하여 이미지에서 범주별 객체를 감지하는 앱을 구축하는 방법을 보여줍니다.

SDK for JavaScript(v3)

와 함께 Amazon Rekognition AWS SDK for JavaScript 을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 이미지에서 Amazon Rekognition을 사용하여 범주별로 객체를 식별하는 앱을 생성하는 방법을 보여줍니다. 이 앱은 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 만듭니다.
- 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

SDK for JavaScript (v3)를 사용한 Amazon SNS 예제

다음 코드 예제에서는 Amazon SNS에서 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)
- [시나리오](#)
- [서버리스 예제](#)

시작하기

Hello Amazon SNS

다음 코드 예제에서는 Amazon SNS 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListTopics](#)를 참조하세요.

작업

CheckIfPhoneNumberIsOptedOut

다음 코드 예시는 CheckIfPhoneNumberIsOptedOut의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CheckIfPhoneNumberIsOptedOut](#)을 참조하세요.

ConfirmSubscription

다음 코드 예시는 ConfirmSubscription의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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:xxxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
// }
return response;
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ConfirmSubscription](#)을 참조하세요.

CreateTopic

다음 코드 예시는 CreateTopic의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateTopic](#)을 참조하세요.

DeleteTopic

다음 코드 예시는 DeleteTopic의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteTopic](#)을 참조하세요.

GetSMSAttributes

다음 코드 예시는 GetSMSAttributes의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetSMSAttributes](#)를 참조하세요.

GetTopicAttributes

다음 코드 예시는 GetTopicAttributes의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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: 'xxxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetTopicAttributes](#)를 참조하세요.

ListSubscriptions

다음 코드 예시는 ListSubscriptions의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListSubscriptions](#)를 참조하세요.

ListTopics

다음 코드 예시는 ListTopics의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListTopics](#)를 참조하세요.

Publish

다음 코드 예시는 Publish의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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-xxxxxxxxxxxxx'
// }
return response;
};
```

그룹, 복제, 속성 옵션을 사용하여 주제에 메시지를 게시하세요.

```
async publishMessages() {
  const message = await this.prompter.input({
    message: MESSAGES.publishMessagePrompt,
  });

  let groupId;
  let deduplicationId;
  let 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();
}
}

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Publish](#)를 참조하세요.

SetSMSAttributes

다음 코드 예시는 SetSMSAttributes의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [SetSMSAttributes](#)를 참조하세요.

SetTopicAttributes

다음 코드 예시는 SetTopicAttributes의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [SetTopicAttributes](#)를 참조하세요.

Subscribe

다음 코드 예시는 Subscribe의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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'
  // }
};
```

모바일 애플리케이션으로 주제 구독.

```
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;
};
```

Lambda 함수에서 주제를 구독합니다.

```
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;
};
```

SQS 대기열로 주제 구독.

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

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
```

```

const command = new SubscribeCommand({
  TopicArn: topicArn,
  Protocol: "sqs",
  Endpoint: queueArn,
});

const response = await client.send(command);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
// }
return response;
};

```

필터를 사용하여 주제를 구독합니다.

```

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

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute set
      // to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({

```

```

        event: ["order_placed"],
      )),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};

```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Subscribe](#)를 참조하세요.

Unsubscribe

다음 코드 예시는 Unsubscribe의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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-xxxxxxxxxxxxxx",
) => {
  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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [Unsubscribe](#)를 참조하세요.

시나리오

DynamoDB 테이블에 데이터를 제출하기 위한 앱 구축

다음 코드 예제에서는 Amazon DynamoDB 테이블에 데이터를 제출하고 사용자가 테이블을 업데이트 하면 알려주는 애플리케이션을 빌드하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SNS

사진을 관리하기 위한 서버리스 애플리케이션 만들기

다음 코드 예시에서는 사용자가 레이블을 사용하여 사진을 관리할 수 있는 서버리스 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 [AWS 커뮤니티](#)의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Amazon Textract 탐색기 애플리케이션 생성

다음 코드 예제에서는 대화형 애플리케이션을 통해 Amazon Textract 출력을 탐색하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript 를 사용하여 Amazon Textract를 사용하여 문서 이미지에서 데이터를 추출하고 대화형 웹 페이지에 표시하는 React 애플리케이션을 빌드하는 방법을 보여줍니다. 이 예제는 웹 브라우저에서 실행되며 자격 증명을 위해 인증된 Amazon Cognito 자격 증명에 필요합니다. 이 애플리케이션은 스토리지로 Amazon Simple Storage Service(Amazon 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

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.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 시나리오의 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";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

위 코드는 필요한 종속성을 제공하고 시나리오를 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.
    const 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;
  let deduplicationId;
  let 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 참조하세요.
 - [CreateQueue](#)

- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Publish](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

API Gateway를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon API Gateway에서 호출한 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

Lambda JavaScript 런타임 API를 사용하여 AWS Lambda 함수를 생성하는 방법을 보여줍니다. 이 예제에서는 다양한 AWS 서비스를 호출하여 특정 사용 사례를 수행합니다. 이 예제에서는 Amazon API Gateway에서 간접 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

예약된 이벤트를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon EventBridge 예약 이벤트에서 호출된 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS Lambda 함수를 호출하는 Amazon EventBridge 예약 이벤트를 생성하는 방법을 보여줍니다. Lambda 함수가 간접 호출될 때 cron 표현식을 사용하여 일정을 예약하도록 EventBridge를 구성합니다. 이 예제에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제는 [AWS SDK for JavaScript v3 개발자 가이드](#)에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

서버리스 예제

Amazon SNS 트리거를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 SNS 주제의 메시지를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 함수는 이벤트 파라미터에서 메시지를 검색하고 각 메시지의 내용을 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 SNS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

TypeScript를 사용하여 Lambda로 SNS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
  }
}
```

```
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

SDK for JavaScript (v3)를 사용한 Amazon SQS 예제

다음 코드 예제에서는 Amazon SQS에서 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [작업](#)
- [시나리오](#)
- [서버리스 예제](#)

시작하기

Hello Amazon SNS

다음 코드 예제에서는 Amazon SQS 사용을 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ListQueues](#)를 참조하세요.

작업

ChangeMessageVisibility

다음 코드 예시는 ChangeMessageVisibility의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ChangeMessageVisibility](#)를 참조하세요.

CreateQueue

다음 코드 예시는 CreateQueue의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateQueue](#)를 참조하세요.

DeleteMessage

다음 코드 예시는 DeleteMessage의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteMessage](#)를 참조하세요.

DeleteMessageBatch

다음 코드 예시는 DeleteMessageBatch의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DeleteMessageBatch](#)를 참조하세요.

DeleteQueue

다음 코드 예시는 DeleteQueue의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteQueue](#)를 참조하세요.

GetQueueAttributes

다음 코드 예시는 GetQueueAttributes의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [GetQueueAttributes](#)를 참조하세요.

GetQueueUrl

다음 코드 예시는 GetQueueUrl의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [GetQueueUrl](#)을 참조하세요.

ListQueues

다음 코드 예시는 ListQueues의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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);
    for (const url of urls) {
      console.log(url);
    }
  }

  return urls;
};
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListQueues](#)를 참조하세요.

ReceiveMessage

다음 코드 예시는 ReceiveMessage의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ReceiveMessage](#)를 참조하세요.

SendMessage

다음 코드 예시는 SendMessage의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [SendMessage](#)를 참조하세요.

SetQueueAttributes

다음 코드 예시는 SetQueueAttributes의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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;
};
```

배달 못한 편지 대기열을 구성합니다.

```
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 참조의 [SetQueueAttributes](#)를 참조하세요.

시나리오

Amazon Textract 탐색기 애플리케이션 생성

다음 코드 예제에서는 대화형 애플리케이션을 통해 Amazon Textract 출력을 탐색하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript 를 사용하여 Amazon Textract를 사용하여 문서 이미지에서 데이터를 추출하고 대화형 웹 페이지에 표시하는 React 애플리케이션을 빌드하는 방법을 보여줍니다. 이 예제는 웹 브라우저에서 실행되며 자격 증명을 위해 인증된 Amazon Cognito 자격 증명에 필요합니다. 이 애플리케이션은 스토리지로 Amazon Simple Storage Service(Amazon 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

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.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 시나리오의 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";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

위 코드는 필요한 종속성을 제공하고 시나리오를 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.
const 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;
  let deduplicationId;
  let 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 참조하세요.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

서버리스 예제

Amazon SQS 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제는 SQS 대기열에서 메시지를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 함수는 이벤트 파라미터에서 메시지를 검색하고 각 메시지의 내용을 로깅합니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 SQS 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0
```

```

exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}

```

TypeScript를 사용하여 Lambda로 SQS 이벤트 사용

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}

```

```

    }
  }
}

```

Amazon SQS 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제는 SQS 대기열에서 이벤트를 수신하는 Lambda 함수에 대한 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda에서 SQS 배치 항목 실패를 보고합니다.

```

// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
  const batchItemFailures = [];
  for (const record of event.Records) {
    try {
      await processMessageAsync(record, context);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }
  return { batchItemFailures };
};

async function processMessageAsync(record, context) {
  if (record.body && record.body.includes("error")) {
    throw new Error("There is an error in the SQS Message.");
  }
  console.log(`Processed message: ${record.body}`);
}

```

TypeScript를 사용하여 Lambda로 SQS 배치 항목 실패를 보고합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
  'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
  Promise<SQSBatchResponse> => {
  const batchItemFailures: SQSBatchItemFailure[] = [];

  for (const record of event.Records) {
    try {
      await processMessageAsync(record);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }

  return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
  if (record.body && record.body.includes("error")) {
    throw new Error('There is an error in the SQS Message.');
```

SDK for JavaScript(v3)를 사용한 Step Functions의 예시

다음 코드 예제에서는 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

StartExecution

다음 코드 예시는 StartExecution의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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:aaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  // }
```

```
// startDate: 2024-01-04T15:54:08.362Z
// }
return response;
}

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfncClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [StartExecution](#)을 참조하세요.

AWS STS SDK for JavaScript(v3)를 사용한 예제

다음 코드 예제에서는 AWS SDK for JavaScript (v3)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS STS.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)

작업

AssumeRole

다음 코드 예시는 AssumeRole의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 수입합니다.

```
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 참조의 [AssumeRole](#)을 참조하세요.

지원 SDK for JavaScript(v3)를 사용한 예제

다음 코드 예제에서는 AWS SDK for JavaScript (v3)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 지원.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)

시작하기

안녕하세요 지원

다음 코드 예제에서는 지원을 사용하여 시작하는 방법을 보여 줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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.",
      );
    }
    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 참조의 [DescribeServices](#)를 참조하세요.

기본 사항

기본 사항 알아보기

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.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 * as inquirer from "@inquirer/prompts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

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.",
      );
    }
    throw err;
  }
};

/**
 * Select a service from the list returned from DescribeServices.
 */
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const selectedService = await inquirer.select({
    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;
};

/**
 * @param {{ categories: import('@aws-sdk/client-support').Category[] }} service
 */
export const getCategory = async (service) => {
  const selectedCategory = await inquirer.select({
    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.select({
      message: "Select a severity level.",
      choices: severityLevels.map((s) => ({ name: s.name, value: s })),
    });
    return selectedSeverityLevel;
  };

/**
 * Create a new support case
 * @param {{
 *   selectedService: import('@aws-sdk/client-support').Service
 *   selectedCategory: import('@aws-sdk/client-support').Category
 *   selectedSeverityLevel: import('@aws-sdk/client-support').SeverityLevel
 * }} selections
 * @returns
 */
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 getTodayOpenCases = 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;
  };

  /**
   * @param {import('@aws-sdk/client-support').Communication[]} communications
   */
```

```

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.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.
 * @param {{
 *   caseId: string,
 *   cases: import('@aws-sdk/client-support').CaseDetails[]
 *   nextToken: string
 * }} options
 * @returns
 */

```



```
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 },
    () => getTodayResolvedCases(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 참조하세요.

- [AddAttachmentsToSet](#)
- [AddCommunicationToCase](#)
- [CreateCase](#)
- [DescribeAttachment](#)
- [DescribeCases](#)
- [DescribeCommunications](#)
- [DescribeServices](#)
- [DescribeSeverityLevels](#)
- [ResolveCase](#)

작업

AddAttachmentsToSet

다음 코드 예시는 AddAttachmentsToSet의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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.
      })
    );
  } catch (err) {
    // Handle error
  }
}
```

```

        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 참조의 [AddAttachmentsToSet](#)를 참조하세요.

AddCommunicationToCase

다음 코드 예시는 AddCommunicationToCase의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [AddCommunicationToCase](#)를 참조하세요.

CreateCase

다음 코드 예시는 CreateCase의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [CreateCase](#)를 참조하세요.

DescribeAttachment

다음 코드 예시는 DescribeAttachment의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeAttachment](#)를 참조하세요.

DescribeCases

다음 코드 예시는 DescribeCases의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeCases](#)를 참조하세요.

DescribeCommunications

다음 코드 예시는 DescribeCommunications의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeCommunications](#)를 참조하세요.

DescribeSeverityLevels

다음 코드 예시는 DescribeSeverityLevels의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [DescribeSeverityLevels](#)를 참조하세요.

ResolveCase

다음 코드 예시는 ResolveCase의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 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 참조의 [ResolveCase](#)를 참조하세요.

SDK for JavaScript(v3)를 사용한 Systems Manager 예제

다음 코드 예제에서는 Systems Manager에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시작하기](#)
- [기본 사항](#)
- [작업](#)

시작하기

Hello Systems Manager

다음 코드 예제에서는 Systems Manager를 사용하여 시작하는 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { paginateListDocuments, SSMClient } from "@aws-sdk/client-ssm";

// Call ListDocuments and display the result.
export const main = async () => {
  const client = new SSMClient();
  const listDocumentsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
```

```
const paginator = paginateListDocuments({ client }, { MaxResults: 5 });
for await (const page of paginator) {
  listDocumentsPaginated.push(...page.DocumentIdentifiers);
}
} catch (caught) {
  console.error(`There was a problem saying hello: ${caught.message}`);
  throw caught;
}

for (const { Name, DocumentFormat, CreatedDate } of listDocumentsPaginated) {
  console.log(`${Name} - ${DocumentFormat} - ${CreatedDate}`);
}
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListDocuments](#)를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 유지 관리 기간을 생성합니다.
- 유지 보수 기간 일정을 수정합니다.
- 문서를 만듭니다.
- 지정된 EC2 인스턴스로 명령을 보냅니다.
- OpsItem을 생성합니다.
- OpsItem을 업데이트하고 해결합니다.
- 유지 보수 기간, OpsItem 및 문서를 삭제합니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { fileURLToPath } from "node:url";
import {
  CreateDocumentCommand,
  CreateMaintenanceWindowCommand,
  CreateOpsItemCommand,
  DeleteDocumentCommand,
  DeleteMaintenanceWindowCommand,
  DeleteOpsItemCommand,
  DescribeOpsItemsCommand,
  DocumentAlreadyExists,
  OpsItemStatus,
  waitUntilCommandExecuted,
  CancelCommandCommand,
  paginateListCommandInvocations,
  SendCommandCommand,
  UpdateMaintenanceWindowCommand,
  UpdateOpsItemCommand,
  SSMClient,
} from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * @typedef {{
 *   ssmClient: import('@aws-sdk/client-ssm').SSMClient,
 *   documentName?: string
 *   maintenanceWindow?: string
 *   winId?: int
 *   ec2InstanceId?: string
```

```
*   requestedDateTime?: Date
*   opsItemId?: string
*   askToDeleteResources?: boolean
* }} State
*/

const defaultMaintenanceWindow = "ssm-maintenance-window";
const defaultDocumentName = "ssmdocument";
// The timeout duration is highly dependent on the specific setup and environment
// necessary. This example handles only the most common error cases, and uses a much
// shorter duration than most productions systems would use.
const COMMAND_TIMEOUT_DURATION_SECONDS = 30; // 30 seconds

const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `Welcome to the AWS Systems Manager SDK Getting Started scenario.
  This program demonstrates how to interact with Systems Manager using the AWS SDK
  for JavaScript V3.
  Systems Manager is the operations hub for your AWS applications and resources
  and a secure end-to-end management solution.
  The program's primary functions include creating a maintenance window, creating
  a document, sending a command to a document,
  listing documents, listing commands, creating an OpsItem, modifying an OpsItem,
  and deleting Systems Manager resources.
  Upon completion of the program, all AWS resources are cleaned up.
  Let's get started...`,
  { header: true },
);

const createMaintenanceWindow = new ScenarioOutput(
  "createMaintenanceWindow",
  "Step 1: Create a Systems Manager maintenance window.",
);

const getMaintenanceWindow = new ScenarioInput(
  "maintenanceWindow",
  "Please enter the maintenance window name:",
  { type: "input", default: defaultMaintenanceWindow },
);
```

```

export const sdkCreateMaintenanceWindow = new ScenarioAction(
  "sdkCreateMaintenanceWindow",
  async (** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new CreateMaintenanceWindowCommand({
          Name: state.maintenanceWindow,
          Schedule: "cron(0 10 ? * MON-FRI *)", //The schedule of the maintenance
window in the form of a cron or rate expression.
          Duration: 2, //The duration of the maintenance window in hours.
          Cutoff: 1, //The number of hours before the end of the maintenance window
that Amazon Web Services Systems Manager stops scheduling new tasks for execution.
          AllowUnassociatedTargets: true, //Allow the maintenance window to run on
managed nodes, even if you haven't registered those nodes as targets.
        })),
      );
      state.winId = response.WindowId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while creating the maintenance window. Please fix the
error and try again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const modifyMaintenanceWindow = new ScenarioOutput(
  "modifyMaintenanceWindow",
  "Modify the maintenance window by changing the schedule.",
);

const sdkModifyMaintenanceWindow = new ScenarioAction(
  "sdkModifyMaintenanceWindow",
  async (** @type {State} */ state) => {
    try {
      await state.ssmClient.send(
        new UpdateMaintenanceWindowCommand({
          WindowId: state.winId,
          Schedule: "cron(0 0 ? * MON *)",
        })),
      );
    } catch (caught) {

```



```

        console.error(caught.message);
        console.log(
            `An error occurred while modifying the maintenance window. Please fix the
            error and try again. Error message: ${caught.message}`,
        );
        throw caught;
    }
},
);

const createSystemsManagerActions = new ScenarioOutput(
    "createSystemsManagerActions",
    "Create a document that defines the actions that Systems Manager performs on your
    EC2 instance.",
);

const getDocumentName = new ScenarioInput(
    "documentName",
    "Please enter the document: ",
    { type: "input", default: defaultDocumentName },
);

const sdkCreateSSMDoc = new ScenarioAction(
    "sdkCreateSSMDoc",
    async (** @type {State} */ state) => {
        const contentData = `{
            "schemaVersion": "2.2",
            "description": "Run a simple shell command",
            "mainSteps": [
                {
                    "action": "aws:runShellScript",
                    "name": "runEchoCommand",
                    "inputs": {
                        "runCommand": [
                            "echo 'Hello, world!'"
                        ]
                    }
                }
            ]
        }`;
        try {
            await state.ssmClient.send(
                new CreateDocumentCommand({
                    Content: contentData,

```

```

        Name: state.documentName,
        DocumentType: "Command",
    })),
    );
} catch (caught) {
    console.log(`Exception type: (${typeof caught})`);
    if (caught instanceof DocumentAlreadyExists) {
        console.log("Document already exists. Continuing...\n");
    } else {
        console.error(caught.message);
        console.log(
            `An error occurred while creating the document. Please fix the error and
            try again. Error message: ${caught.message}`,
        );
        throw caught;
    }
},
);

const ec2HelloWorld = new ScenarioOutput(
    "ec2HelloWorld",
    `Now you have the option of running a command on an EC2 instance that echoes
    'Hello, world!'. In order to run this command, you must provide the instance ID
    of a Linux EC2 instance. If you do not already have a running Linux EC2 instance
    in your account, you can create one using the AWS console. For information about
    creating an EC2 instance, see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/
    ec2-launch-instance-wizard.html.`,
);

const enterIdOrSkipEC2HelloWorld = new ScenarioInput(
    "enterIdOrSkipEC2HelloWorld",
    "Enter your EC2 InstanceId or press enter to skip this step: ",
    { type: "input", default: "" },
);

const sdkEC2HelloWorld = new ScenarioAction(
    "sdkEC2HelloWorld",
    async (/** @type {State} */ state) => {
        try {
            const response = await state.ssmClient.send(
                new SendCommandCommand({
                    DocumentName: state.documentName,
                    InstanceIds: [state.ec2InstanceId],
                })
            );
        } catch (error) {
            console.error("Error sending command:", error);
        }
    })
);
```

```

        TimeoutSeconds: COMMAND_TIMEOUT_DURATION_SECONDS,
      )),
    );
    state.CommandId = response.Command.CommandId;
  } catch (caught) {
    console.error(caught.message);
    console.log(
      `An error occurred while sending the command. Please fix the error and try
again. Error message: ${caught.message}`,
    );
    throw caught;
  }
},
{
  skipWhen: (/** @type {State} */ state) =>
    state.enterIdOrSkipEC2HelloWorld === "",
},
);

const sdkGetCommandTime = new ScenarioAction(
  "sdkGetCommandTime",
  async (/** @type {State} */ state) => {
    const listInvocationsPaginated = [];
    console.log(
      "Let's get the time when the specific command was sent to the specific managed
node.",
    );

    console.log(
      `First, we'll wait for the command to finish executing. This may take up to
${COMMAND_TIMEOUT_DURATION_SECONDS} seconds.`,
    );
    const commandExecutedResult = waitUntilCommandExecuted(
      { client: state.ssmClient },
      {
        CommandId: state.CommandId,
        InstanceId: state.ec2InstanceId,
      },
    );
    // This is necessary because the TimeoutSeconds of SendCommandCommand is only
    for the delivery, not execution.
    try {
      await new Promise((_, reject) =>
        setTimeout(

```

```

        reject,
        COMMAND_TIMEOUT_DURATION_SECONDS * 1000,
        new Error("Command Timed Out"),
    ),
);
} catch (caught) {
    if (caught.message === "Command Timed Out") {
        commandExecutedResult.state = "TIMED_OUT";
    } else {
        throw caught;
    }
}

if (commandExecutedResult.state !== "SUCCESS") {
    console.log(
        `The command with id: ${state.CommandId} did not execute in the allotted
time. Canceling command.`,
    );
    state.ssmClient.send(
        new CancelCommandCommand({
            CommandId: state.CommandId,
        }),
    );
    state.enterIdOrSkipEC2HelloWorld === "";
    return;
}

for await (const page of paginateListCommandInvocations(
    { client: state.ssmClient },
    { CommandId: state.CommandId },
)) {
    listInvocationsPaginated.push(...page.CommandInvocations);
}
/**
 * @type {import('@aws-sdk/client-ssm').CommandInvocation}
 */
const commandInvocation = listInvocationsPaginated.shift(); // Because the call
was made with CommandId, there's only one result, so shift it off.
state.requestedDateTime = commandInvocation.RequestedDateTime;

console.log(
    `The command invocation happened at: ${state.requestedDateTime}.`,
);
},

```

```

{
  skipWhen: (/** @type {State} */ state) =>
    state.enterIdOrSkipEC2HelloWorld === "",
},
);

const createSSMOpsItem = new ScenarioOutput(
  "createSSMOpsItem",
  `Now we will create a Systems Manager OpsItem. An OpsItem is a feature provided by
the Systems Manager service. It is a type of operational data item that allows you
to manage and track various operational issues, events, or tasks within your AWS
environment.
You can create OpsItems to track and manage operational issues as they arise. For
example, you could create an OpsItem whenever your application detects a critical
error or an anomaly in your infrastructure.`
);

const sdkCreateSSMOpsItem = new ScenarioAction(
  "sdkCreateSSMOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new CreateOpsItemCommand({
          Description: "Created by the System Manager Javascript API",
          Title: "Disk Space Alert",
          Source: "EC2",
          Category: "Performance",
          Severity: "2",
        })
      );
      state.opsItemId = response.OpsItemId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while creating the ops item. Please fix the error and try
again. Error message: ${caught.message}`
      );
      throw caught;
    }
  },
);

const updateOpsItem = new ScenarioOutput(
  "updateOpsItem",

```

```
(/** @type {State} */ state) =>
  `Now we will update the OpsItem: ${state.opsItemId}`,
);

const sdkUpdateOpsItem = new ScenarioAction(
  "sdkUpdateOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const _response = await state.ssmClient.send(
        new UpdateOpsItemCommand({
          OpsItemId: state.opsItemId,
          Description: `An update to ${state.opsItemId}`,
        }),
      );
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while updating the ops item. Please fix the error and try
again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const getOpsItemStatus = new ScenarioOutput(
  "getOpsItemStatus",
  (/** @type {State} */ state) =>
    `Now we will get the status of the OpsItem: ${state.opsItemId}`,
);

const sdkOpsItemStatus = new ScenarioAction(
  "sdkGetOpsItemStatus",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new DescribeOpsItemsCommand({
          OpsItemId: state.opsItemId,
        }),
      );
      state.opsItemStatus = response.OpsItemStatus;
    } catch (caught) {
      console.error(caught.message);
      console.log(
```

```
        `An error occurred while describing the ops item. Please fix the error and
        try again. Error message: ${caught.message}`,
        );
        throw caught;
    }
},
);

const resolveOpsItem = new ScenarioOutput(
    "resolveOpsItem",
    (/** @type {State} */ state) =>
        `Now we will resolve the OpsItem: ${state.opsItemId}`,
);

const sdkResolveOpsItem = new ScenarioAction(
    "sdkResolveOpsItem",
    async (/** @type {State} */ state) => {
        try {
            const _response = await state.ssmClient.send(
                new UpdateOpsItemCommand({
                    OpsItemId: state.opsItemId,
                    Status: OpsItemStatus.RESOLVED,
                }),
            );
        } catch (caught) {
            console.error(caught.message);
            console.log(
                `An error occurred while updating the ops item. Please fix the error and try
                again. Error message: ${caught.message}`,
            );
            throw caught;
        }
    },
);

const askToDeleteResources = new ScenarioInput(
    "askToDeleteResources",
    "Would you like to delete the Systems Manager resources created during this
    example run?",
    { type: "confirm" },
);

const confirmDeleteChoice = new ScenarioOutput(
    "confirmDeleteChoice",
```

```
(/** @type {State} */ state) => {
  if (state.askToDeleteResources) {
    return "You chose to delete the resources.";
  }
  return "The Systems Manager resources will not be deleted. Please delete them manually to avoid charges.";
},
);

export const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    try {
      await state.ssmClient.send(
        new DeleteOpsItemCommand({
          OpsItemId: state.opsItemId,
        }),
      );
      console.log(`The ops item: ${state.opsItemId} was successfully deleted.`);
    } catch (caught) {
      console.log(
        `There was a problem deleting the ops item: ${state.opsItemId}. Please delete it manually. Error: ${caught.message}`,
      );
    }

    try {
      await state.ssmClient.send(
        new DeleteMaintenanceWindowCommand({
          Name: state.maintenanceWindow,
          WindowId: state.winId,
        }),
      );
      console.log(
        `The maintenance window: ${state.maintenanceWindow} was successfully deleted.`,
      );
    } catch (caught) {
      console.log(
        `There was a problem deleting the maintenance window: ${state.opsItemId}. Please delete it manually. Error: ${caught.message}`,
      );
    }
  }
);
```



```
    try {
      await state.ssmClient.send(
        new DeleteDocumentCommand({
          Name: state.documentName,
        }),
      );
      console.log(
        `The document: ${state.documentName} was successfully deleted.`
      );
    } catch (caught) {
      console.log(
        `There was a problem deleting the document: ${state.documentName}. Please delete it manually. Error: ${caught.message}`
      );
    }
  },
  { skipWhen: (/** @type {} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the Systems Manager Basics scenario for the AWS Javascript SDK v3. Thank you!",
);

const myScenario = new Scenario(
  "SSM Basics",
  [
    greet,
    pressEnter,
    createMaintenanceWindow,
    getMaintenanceWindow,
    sdkCreateMaintenanceWindow,
    modifyMaintenanceWindow,
    pressEnter,
    sdkModifyMaintenanceWindow,
    createSystemsManagerActions,
    getDocumentName,
    sdkCreateSSMDoc,
    ec2HelloWorld,
    enterIdOrSkipEC2HelloWorld,
    sdkEC2HelloWorld,
    sdkGetCommandTime,
    pressEnter,
```

```

    createSSMOpsItem,
    pressEnter,
    sdkCreateSSMOpsItem,
    updateOpsItem,
    pressEnter,
    sdkUpdateOpsItem,
    getOpsItemStatus,
    pressEnter,
    sdkOpsItemStatus,
    resolveOpsItem,
    pressEnter,
    sdkResolveOpsItem,
    askToDeleteResources,
    confirmDeleteChoice,
    sdkDeleteResources,
    goodbye,
  ],
  { ssmClient: new SSMClient({}) },
);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
 - [CreateDocument](#)
 - [CreateMaintenanceWindow](#)

- [CreateOpsItem](#)
- [DeleteMaintenanceWindow](#)
- [ListCommandInvocations](#)
- [SendCommand](#)
- [UpdateOpsItem](#)

작업

CreateDocument

다음 코드 예시는 CreateDocument의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { CreateDocumentCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ content, name, documentType }) => {
  const client = new SSMClient({});
  try {
    const { documentDescription } = await client.send(
      new CreateDocumentCommand({
        Content: content, // The content for the new SSM document. The content must
        not exceed 64KB.
        Name: name,
        DocumentType: documentType, // Document format type can be JSON, YAML, or
        TEXT. The default format is JSON.
      })
    );
  }
};
```

```

    console.log("Document created successfully.");
    return { DocumentDescription: documentDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DocumentAlreadyExists") {
      console.warn(`${caught.message}. Did you provide a new document name?`);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateDocument](#)를 참조하세요.

CreateMaintenanceWindow

다음 코드 예시는 CreateMaintenanceWindow의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { CreateMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM maintenance window.
 * @param {{ name: string, allowUnassociatedTargets: boolean, duration: number,
 * cutoff: number, schedule: string, description?: string }}
 */
export const main = async ({
  name,
  allowUnassociatedTargets, // Allow the maintenance window to run on managed nodes,
  even if you haven't registered those nodes as targets.
  duration, // The duration of the maintenance window in hours.
  cutoff, // The number of hours before the end of the maintenance window that
  Amazon Web Services Systems Manager stops scheduling new tasks for execution.

```

```

    schedule, // The schedule of the maintenance window in the form of a cron or rate
    expression.
    description = undefined,
  }) => {
    const client = new SSMClient({});

    try {
      const { windowId } = await client.send(
        new CreateMaintenanceWindowCommand({
          Name: name,
          Description: description,
          AllowUnassociatedTargets: allowUnassociatedTargets, // Allow the maintenance
window to run on managed nodes, even if you haven't registered those nodes as
targets.
          Duration: duration, // The duration of the maintenance window in hours.
          Cutoff: cutoff, // The number of hours before the end of the maintenance
window that Amazon Web Services Systems Manager stops scheduling new tasks for
execution.
          Schedule: schedule, // The schedule of the maintenance window in the form of
a cron or rate expression.
        })),
      );
      console.log(`Maintenance window created with Id: ${windowId}`);
      return { WindowId: windowId };
    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        console.warn(`${caught.message}. Did you provide these values?`);
      } else {
        throw caught;
      }
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateMaintenanceWindow](#)를 참조하세요.

CreateOpsItem

다음 코드 예시는 CreateOpsItem의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { CreateOpsItemCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM OpsItem.
 * @param {{ title: string, source: string, category?: string, severity?: string }}
 */
export const main = async ({
  title,
  source,
  category = undefined,
  severity = undefined,
}) => {
  const client = new SSMClient({});
  try {
    const { opsItemArn, opsItemId } = await client.send(
      new CreateOpsItemCommand({
        Title: title,
        Source: source, // The origin of the OpsItem, such as Amazon EC2 or Systems
        Category: category,
        Severity: severity,
      })
    );
    console.log(`Ops item created with id: ${opsItemId}`);
    return { OpsItemArn: opsItemArn, OpsItemId: opsItemId };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [CreateOpsItem](#)을 참조하세요.

DeleteDocument

다음 코드 예시는 DeleteDocument의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DeleteDocumentCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Delete an SSM document.
 * @param {{ documentName: string }}
 */
export const main = async ({ documentName }) => {
  const client = new SSMClient({});
  try {
    await client.send(new DeleteDocumentCommand({ Name: documentName }));
    console.log(`Document '${documentName}' deleted.`);
    return { Deleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteDocument](#)를 참조하세요.

DeleteMaintenanceWindow

다음 코드 예시는 DeleteMaintenanceWindow의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DeleteMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Delete an SSM maintenance window.
 * @param {{ windowId: string }}
 */
export const main = async ({ windowId }) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new DeleteMaintenanceWindowCommand({ WindowId: windowId }),
    );
    console.log(`Maintenance window '${windowId}' deleted.`);
    return { Deleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    } else {
      throw caught;
    }
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteMaintenanceWindow](#)를 참조하세요.

DescribeOpsItems

다음 코드 예시는 DescribeOpsItems의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  OpsItemFilterOperator,
  OpsItemFilterKey,
  paginateDescribeOpsItems,
  SSMClient,
} from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Describe SSM OpsItems.
 * @param {{ opsItemId: string }}
 */
export const main = async ({ opsItemId }) => {
  const client = new SSMClient({});
  try {
    const describeOpsItemsPaginated = [];
    for await (const page of paginateDescribeOpsItems(
      { client },
      {
        OpsItemFilters: {
          Key: OpsItemFilterKey.OPSITEM_ID,
          Operator: OpsItemFilterOperator.EQUAL,
          Values: opsItemId,
        },
      },
    )) {
      describeOpsItemsPaginated.push(...page.OpsItemSummaries);
    }
    console.log("Here are the ops items:");
    console.log(describeOpsItemsPaginated);
    return { OpsItemSummaries: describeOpsItemsPaginated };
  }
}
```

```

    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        console.warn(`${caught.message}. Did you provide this value?`);
      }
      throw caught;
    }
  };
};

```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [DescribeOpsItems](#)를 참조하세요.

ListCommandInvocations

다음 코드 예시는 ListCommandInvocations의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { paginateListCommandInvocations, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * List SSM command invocations on an instance.
 * @param {{ instanceId: string }}
 */
export const main = async ({ instanceId }) => {
  const client = new SSMClient({});
  try {
    const listCommandInvocationsPaginated = [];
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListCommandInvocations(
      { client },
      {
        InstanceId: instanceId,
      },
    );
    for await (const page of paginator) {

```

```

        listCommandInvocationsPaginated.push(...page.CommandInvocations);
    }
    console.log("Here is the list of command invocations:");
    console.log(listCommandInvocationsPaginated);
    return { CommandInvocations: listCommandInvocationsPaginated };
} catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
        console.warn(`${caught.message}. Did you provide a valid instance ID?`);
    }
    throw caught;
}
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListCommandInvocations](#)를 참조하세요.

SendCommand

다음 코드 예시는 SendCommand의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { SendCommandCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Send an SSM command to a managed node.
 * @param {{ documentName: string }}
 */
export const main = async ({ documentName }) => {
    const client = new SSMClient({});
    try {
        await client.send(
            new SendCommandCommand({
                DocumentName: documentName,
            }),

```

```

    );
    console.log("Command sent successfully.");
    return { Success: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
      console.warn(`${caught.message}. Did you provide a valid document name?`);
    } else {
      throw caught;
    }
  }
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [SendCommand](#)를 참조하세요.

UpdateMaintenanceWindow

다음 코드 예시는 UpdateMaintenanceWindow의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```

import { UpdateMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Update an SSM maintenance window.
 * @param {{ windowId: string, allowUnassociatedTargets?: boolean, duration?:
 * number, enabled?: boolean, name?: string, schedule?: string }}
 */
export const main = async ({
  windowId,
  allowUnassociatedTargets = undefined, //Allow the maintenance window to run on
  managed nodes, even if you haven't registered those nodes as targets.
  duration = undefined, //The duration of the maintenance window in hours.
  enabled = undefined,

```

```

    name = undefined,
    schedule = undefined, //The schedule of the maintenance window in the form of a
    cron or rate expression.
  }) => {
    const client = new SSMClient({});
    try {
      const { opsItemArn, opsItemId } = await client.send(
        new UpdateMaintenanceWindowCommand({
          WindowId: windowId,
          AllowUnassociatedTargets: allowUnassociatedTargets,
          Duration: duration,
          Enabled: enabled,
          Name: name,
          Schedule: schedule,
        }),
      );
      console.log("Maintenance window updated.");
      return { OpsItemArn: opsItemArn, OpsItemId: opsItemId };
    } catch (caught) {
      if (caught instanceof Error && caught.name === "ValidationError") {
        console.warn(`${caught.message}. Are these values correct?`);
      } else {
        throw caught;
      }
    }
  };
};

```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateMaintenanceWindow](#)를 참조하세요.

UpdateOpsItem

다음 코드 예시는 UpdateOpsItem의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { UpdateOpsItemCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Update an SSM OpsItem.
 * @param {{ opsItemId: string, status?: OpsItemStatus }}
 */
export const main = async ({
  opsItemId,
  status = undefined, // The OpsItem status. Status can be Open, In Progress, or Resolved
}) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new UpdateOpsItemCommand({
        OpsItemId: opsItemId,
        Status: status,
      }),
    );
    console.log("Ops item updated.");
    return { Success: true };
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "OpsItemLimitExceededException"
    ) {
      console.warn(
        `Couldn't create ops item because you have exceeded your open OpsItem limit. ${caught.message}.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 [UpdateOpsItem](#)을 참조하세요.

SDK for JavaScript(v3)를 사용한 Amazon Textract 예제

다음 코드 예제에서는 Amazon Textract에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

Amazon Textract 탐색기 애플리케이션 생성

다음 코드 예제에서는 대화형 애플리케이션을 통해 Amazon Textract 출력을 탐색하는 방법을 보여줍니다.

SDK for JavaScript(v3)

AWS SDK for JavaScript 를 사용하여 Amazon Textract를 사용하여 문서 이미지에서 데이터를 추출하고 대화형 웹 페이지에 표시하는 React 애플리케이션을 빌드하는 방법을 보여줍니다. 이 예제는 웹 브라우저에서 실행되며 자격 증명을 위해 인증된 Amazon Cognito 자격 증명が必要です. 이 애플리케이션은 스토리지로 Amazon Simple Storage Service(Amazon 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 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

고객 피드백 분석을 위한 애플리케이션 생성

다음 코드 예제에서는 고객 의견 카드를 분석하고, 원어에서 번역하고, 감정을 파악하고, 번역된 텍스트에서 오디오 파일을 생성하는 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 코드와 배포 지침은 [GitHub](#)의 프로젝트를 참조하세요. 다음 발췌문은 Lambda 함수 내에서 AWS SDK for JavaScript가 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 예제에서 사용되는 서비스

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

SDK for JavaScript (v3)를 사용한 Amazon Transcribe 예

다음 코드 예제에서는 Amazon Transcribe에서 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 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [작업](#)
- [시나리오](#)

작업

DeleteMedicalTranscriptionJob

다음 코드 예시는 DeleteMedicalTranscriptionJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
import { TranscribeClient } from "@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 전사 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_transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 자세한 정보는 [AWS SDK for JavaScript 개발자 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteMedicalTranscriptionJob](#)을 참조하세요.

DeleteTranscriptionJob

다음 코드 예시는 DeleteTranscriptionJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

전사 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, 'transcription_demo'
};

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

run();
```

클라이언트를 생성합니다.

```
import { TranscribeClient } from "@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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [DeleteTranscriptionJob](#)을 참조하세요.

ListMedicalTranscriptionJobs

다음 코드 예시는 ListMedicalTranscriptionJobs의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
import { TranscribeClient } from "@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 전사 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListMedicalTranscriptionJobs](#)를 참조하세요.

ListTranscriptionJobs

다음 코드 예시는 ListTranscriptionJobs의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

전사 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.

```
import { TranscribeClient } from "@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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [ListTranscriptionJobs](#)를 참조하세요.

StartMedicalTranscriptionJob

다음 코드 예시는 StartMedicalTranscriptionJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

 Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

클라이언트를 생성합니다.

```
import { TranscribeClient } from "@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 전사 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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [StartMedicalTranscriptionJob](#)을 참조하세요.

StartTranscriptionJob

다음 코드 예시는 StartTranscriptionJob의 사용 방법을 보여줍니다.

SDK for JavaScript (v3)

Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예 리포지토리](#)에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

전사 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.

```

import { TranscribeClient } from "@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 안내서](#)를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 [StartTranscriptionJob](#)을 참조하세요.

시나리오

Amazon Transcribe 스트리밍 앱 구축

다음 코드 예제에서는 라이브 오디오를 실시간으로 기록, 변환 및 번역하고 결과를 이메일로 보내는 앱을 구축하는 방법을 보여줍니다.

SDK for JavaScript (v3)

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 [GitHub](#)에서 전체 예시를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

SDK for JavaScript(v3)를 사용한 Amazon Translate 예제

다음 코드 예제에서는 Amazon Translate에서 AWS SDK for JavaScript (v3)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- [시나리오](#)

시나리오

Amazon Transcribe 스트리밍 앱 구축

다음 코드 예제에서는 라이브 오디오를 실시간으로 기록, 변환 및 번역하고 결과를 이메일로 보내는 앱을 구축하는 방법을 보여줍니다.

SDK for JavaScript (v3)

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 [GitHub](#)에서 전체 예시를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Amazon Lex 챗봇 구축

다음 코드 예제에서는 챗봇을 만들어 웹사이트 방문자를 참여시키는 방법을 보여줍니다.

SDK for JavaScript(v3)

Amazon Lex API를 사용하여 웹 애플리케이션 내에 챗봇을 구축하여 웹 사이트 방문자의 참여를 유도하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 AWS SDK for JavaScript 개발자 안내서의 [Amazon Lex 챗봇 구축](#) 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

고객 피드백 분석을 위한 애플리케이션 생성

다음 코드 예제에서는 고객 의견 카드를 분석하고, 원어에서 번역하고, 감정을 파악하고, 번역된 텍스트에서 오디오 파일을 생성하는 애플리케이션을 생성하는 방법을 보여줍니다.

SDK for JavaScript(v3)

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 코드와 배포 지침은 [GitHub](#)의 프로젝트를 참조하세요. 다음 발췌문은 Lambda 함수 내에서 AWS SDK for JavaScript가 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 예제에서 사용되는 서비스

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

이 AWS 제품 또는 서비스에 대한 보안

Amazon Web Services(AWS)에서 가장 우선순위가 높은 것이 클라우드 보안입니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다. 보안은 AWS와 사용자 간의 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

클라우드 보안 - AWS는 클라우드에서 제공되는 모든 서비스를 실행하는 인프라를 보호하고 안전하게 사용할 수 있는 서비스를 AWS 제공할 책임이 있습니다. 당사의 보안 책임은에서 최우선 순위이며 AWS, 타사 감사자는 [AWS 규정 준수 프로그램의](#) 일환으로 보안의 효과를 정기적으로 테스트하고 검증합니다.

클라우드의 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 모델](#)을 따릅니다. AWS 서비스 보안 정보는 [AWS 서비스 보안 설명서 페이지](#) 및 규정 [AWSAWS 준수 프로그램의 규정 준수 작업 범위에 속하는 서비스를 참조하세요](#).

주제

- [이 AWS 제품 또는 서비스의 데이터 보호](#)
- [자격 증명 및 액세스 관리](#)
- [이 AWS 제품 또는 서비스에 대한 규정 준수 검증](#)
- [이 AWS 제품 또는 서비스에 대한 복원력](#)
- [이 AWS 제품 또는 서비스에 대한 인프라 보안](#)
- [최소 TLS 버전 적용](#)

이 AWS 제품 또는 서비스의 데이터 보호

AWS [공동 책임 모델](#)이 AWS 제품 또는 서비스의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 사용하는 AWS 서비스의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 관한 자세한 내용은 [데이터 프라이버시 FAQ](#)를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 [AWS 공동 책임 모델 및 GDPR](#) 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 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 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 [CloudTrail 추적 작업을 참조하세요](#).
- 내부의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 [연방 정보 처리 표준\(FIPS\) 140-3](#)을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 이 AWS 제품 또는 서비스 또는 기타 AWS 서비스에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 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.

자격 증명 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 되는 AWS 서비스입니다. IAM 관리자는 누가 AWS 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제

- [대상](#)
- [ID를 통한 인증](#)
- [정책을 사용하여 액세스 관리](#)

- [IAM AWS 서비스 작업 방법](#)
- [AWS 자격 증명 및 액세스 문제 해결](#)

대상

AWS Identity and Access Management (IAM)를 사용하는 방법에서 수행하는 작업에 따라 다릅니다 AWS.

서비스 사용자 - AWS 서비스 를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 AWS 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. 에서 기능에 액세스할 수 없는 경우 사용 증인의 [AWS 자격 증명 및 액세스 문제 해결](#) 또는 사용 설명서를 AWS참조 AWS 서비스 하세요.

서비스 관리자 - 회사에서 AWS 리소스를 책임지고 있는 경우에 대한 전체 액세스 권한을 가지고 있을 것입니다 AWS. 서비스 관리자는 서비스 사용자가 액세스해야 하는 AWS 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 IAM을 사용하는 방법에 대해 자세히 알아보려면 사용 증인의 AWS 서비스 사용 설명서를 AWS참조하세요.

IAM 관리자 - IAM 관리자라면 AWS에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 자격 AWS 증명 기반 정책 예제를 보려면 사용 증인의 사용 설명서를 참조 AWS 서비스 하세요.

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증되어야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 [AWS 계정에 로그인하는 방법](#) 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해서는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 [API 요청용AWS Signature Version 4](#) 섹션을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명이 필요한 작업은 IAM 사용 설명서의 [루트 사용자 자격 증명이 필요한 작업](#) 섹션을 참조하세요.

페더레이션 ID

가장 좋은 방법은 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 서비스 사용하여 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 디렉터리, 웹 자격 증명 공급자 또는 자격 증명 소스의 자격 증명을 AWS 서비스 사용하여 Directory Service 에 액세스하는 사용자입니다. 페더레이션 ID는 임시 자격 증명을 제공하는 역할을 수임합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center를 추천합니다. 자세한 정보는 AWS IAM Identity Center 사용 설명서의 [What is IAM Identity Center?](#)를 참조하세요.

IAM 사용자 및 그룹

[IAM 사용자](#)는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명에 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여 액세스하도록 인간 사용자에게 요구](#)하기를 참조하세요.

[IAM 그룹](#)은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자 사용 사례](#) 섹션을 참조하세요.

IAM 역할

[IAM 역할](#)은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. [사용자에서 IAM 역할\(콘솔\)로 전환하거나 또는 API 작업을 호출하여 역할을](#) 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 [역할 수임 방법](#)을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 [교차 계정 리소스 액세스](#)를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다. 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 [JSON 정책 개요](#) 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

ID 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 [고객 관리형 정책으로 사용자 지정 IAM 권한 정의](#)를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책 및 인라인 정책 중에서 선택](#) 섹션을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF 및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 [액세스 제어 목록\(ACL\) 개요](#)를 참조하세요.

기타 정책 타입

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 [IAM 엔터티의 권한 범위](#)를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 [서비스 제어 정책](#)을 참조하세요.
- 리소스 제어 정책(RCP) - 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 [리소스 제어 정책\(RCP\)](#)을 참조하세요.
- 세션 정책 - 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 [세션 정책](#)을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 참조하세요.

IAM AWS 서비스 작업 방법

가 대부분의 IAM 기능을 AWS 서비스 사용하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 [AWS IAM으로 작업하는 서비스를](#) 참조하세요.

IAM AWS 서비스 에서 특징을 사용하는 방법을 알아보려면 관련 서비스 사용 설명서의 보안 섹션을 참조하세요.

AWS 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단 AWS 하고 수정할 수 있습니다.

주제

- [에서 작업을 수행할 권한이 없음 AWS](#)

- [iam:PassRole을 수행하도록 인증되지 않음](#)
- [내 외부의 사람이 내 AWS 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.](#)

에서 작업을 수행할 권한이 없음 AWS

작업을 수행할 권한이 없다는 오류가 표시되면 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 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 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 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 권한이 없습니다.

```
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 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 AWS 리소스에 액세스 AWS 계정 하도록 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 제

어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세한 내용은 다음을 참조하세요.

- 에서 이러한 기능을 AWS 지원하는지 여부를 알아보려면 섹션을 참조하세요 [IAM AWS 서비스 작업 방법](#).
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 [IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요](#).
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [타사가 AWS 계정 소유한에 대한 액세스 권한 제공을](#) AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [외부에서 인증된 사용자에게 액세스 권한 제공\(ID 페더레이션\)](#)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 [IAM의 크로스 계정 리소스 액세스](#)를 참조하세요.

이 AWS 제품 또는 서비스에 대한 규정 준수 검증

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 [AWS 서비스 규정 준수 프로그램 제공 범위 내](#)를 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 [AWS 규정 준수 프로그램](#).

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 [Downloading Reports inDownloading AWS Artifact](#)을 참조하세요.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 AWS 서비스 결정됩니다. 사용 시 규정 준수 책임에 대한 자세한 내용은 [AWS 보안 설명서를](#) AWS 서비스참조하세요.

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 규정 [AWSAWS 준수 프로그램의 규정 준수 작업 범위에 속하는 서비스를](#) 참조하세요.

이 AWS 제품 또는 서비스에 대한 복원력

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다.

AWS 리전 는 지연 시간이 짧고 처리량이 많으며 중복성이 높은 네트워킹과 연결된 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다.

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 리전 및 가용 영역에 대한 자세한 내용은 [AWS 글로벌 인프라](#)를 참조하세요.

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 규정 [AWSAWS 준수 프로그램의 규정 준수 작업 범위에 속하는 서비스](#)를 참조하세요.

이 AWS 제품 또는 서비스에 대한 인프라 보안

이 AWS 제품 또는 서비스는 관리형 서비스를 사용하므로 글로벌 네트워크 보안으로 AWS 보호됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 [AWS 클라우드 보안](#)을 참조하세요. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 [인프라 보호](#)를 참조하세요.

AWS 에서 게시한 API 호출을 사용하여 네트워크를 통해이 AWS 제품 또는 서비스에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- Transport Layer Security(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 위탁자와 관련된 시크릿 액세스 키를 사용하여 서명해야 합니다. 또는 [AWS Security Token Service](#)(AWS STS)를 사용하여 임시 보안 자격 증명을 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 규정 [AWSAWS 준수 프로그램의 규정 준수 작업 범위에 속하는 서비스](#)를 참조하세요.

최소 TLS 버전 적용

AWS 서비스와 통신할 때 보안을 강화하려면 TLS 1.2 이상을 사용하도록 AWS SDK for JavaScript 를 구성합니다.

전송 계층 보안(TLS)은 웹 브라우저 및 기타 애플리케이션에서 네트워크를 통해 교환되는 데이터의 프라이버시 및 무결성을 보장하기 위해 사용하는 프로토콜입니다.

Important

2024년 6월 10일부터 각 AWS 리전의 AWS 서비스 API 엔드포인트에서 TLS 1.3을 사용할 수 있다고 [발표했습니다](#). AWS SDK for JavaScript v3는 TLS 버전 자체를 협상하지 않습니다. 대신, `https.Agent`를 통해 구성할 수 있는 Node.js에서 결정된 TLS 버전을 사용합니다. AWS에서는 Node.js의 현재 활성 LTS 버전을 사용할 것을 권장합니다.

Node.js에서 TLS 확인 및 적용

Node.js와 AWS SDK for JavaScript 함께를 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. 사용 가능한 경우 Node.js는 기본적으로 TLS 1.3을 사용합니다. 필요한 경우 다른 버전을 명시적으로 지정할 수 있습니다.

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.

지원되는 최소 및 최대 TLS 버전 확인

개발자는 다음 스크립트를 사용하여 Node.js에서 지원되는 최소 및 최대 TLS 버전을 확인할 수 있습니다.

```
import tls from "tls";  
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +  
  tls.DEFAULT_MAX_VERSION);
```

마지막 명령은 다음 예제와 같이 기본값 최소 및 최대 TLS 버전을 출력합니다.

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

TLS의 최소 버전 적용

Node.js는 호출이 실패하면 TLS 버전을 협상합니다. 명령줄에서 스크립트를 실행할 때 또는 JavaScript 코드의 요청에 따라 이 협상 중에 허용 가능한 최소 TLS 버전을 적용할 수 있습니다.

명령줄에서 최소 TLS 버전을 지정하려면 Node.js 버전 11.4.0 이상을 사용해야 합니다. 특정 Node.js 버전을 설치하려면 먼저, [Node version manager installing and updating](#)에 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 대해 허용 가능한 최소 TLS 버전을 지정하려면 다음 예제와 같이 `minVersion` 파라미터를 사용하여 프로토콜을 지정합니다.

```
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({
      {
        minVersion: 'TLSv1.2'
      }
    })
  })
});
```

Enforce TLS 1.3

TLS 1.3이 허용 가능한 최소 버전인 경우 이를 적용하려면 다음 예와 같이 스크립트를 실행할 때 `--tls-min-v1.3` 인수를 지정합니다.

```
node --tls-min-v1.3 yourScript.js
```

JavaScript 코드에서 특정 요청에 대해 허용 가능한 최소 TLS 버전을 지정하려면 다음 예제와 같이 `minVersion` 파라미터를 사용하여 프로토콜을 지정합니다.

```
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({
      {
        minVersion: 'TLSv1.3'
      }
    })
  })
});
```

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

브라우저 스크립트에서 TLS 확인 및 적용

브라우저 스크립트에서 SDK for JavaScript를 사용하면 브라우저 설정이 사용되는 TLS 버전을 제어합니다. 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.

AWS SDK for JavaScript v3 요청에서 TLS 버전 검색

다음 스크립트를 사용하여 AWS SDK 요청에 사용되는 TLS 버전을 로깅할 수 있습니다.

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

const client = new S3Client({ region: "us-east-1" });

const tlsSocket = new tls.TLSSocket();

client.middlewareStack.add((next, context) => async (args) => {
  console.log(`Using TLS version: ${tlsSocket.getProtocol()}`);
  return next(args);
});
```

마지막 명령은 다음 예제와 같이 사용 중인 TLS 버전을 출력합니다.

```
Using TLS version: TLSv1.3
```


버전 2.x에서 3.x로 마이그레이션 AWS SDK for JavaScript

AWS SDK for JavaScript 버전 3은 버전 2의 주요 재작성입니다. 이 섹션에서는 두 버전의 차이점을 설명하고 SDK for JavaScript 버전 2에서 버전 3으로 마이그레이션하는 방법을 설명합니다.

codemod를 사용하여 SDK for JavaScript v3로 코드 마이그레이션

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. v2에서 변경된 내용과 각 변경 사항에 대한 v3는 [AWS SDK for JavaScript GitHub 리포지토리의 마이그레이션 가이드에서 확인할 수 있습니다](#).

AWS SDK for JavaScript v3를 최대한 활용하려면 아래 설명된 codemod 스크립트를 사용하는 것이 좋습니다.

codemod를 사용하여 기존 v2 코드 마이그레이션

[aws-sdk-js-codemod](#)의 codemod 스크립트 모음은 v3 APIs를 사용하도록 기존 AWS SDK for JavaScript (v2) 애플리케이션을 마이그레이션하는 데 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 });
await client.listTables({}).promise()
  .then(console.log)
  .catch(console.error);
```

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 });
await client.listTables({})
  .then(console.log)
  .catch(console.error);
```

일반적인 사용 사례에 대한 변환을 구현했습니다. 코드가 올바르게 변환되지 않는 경우 입력 코드 예와 관찰/예상된 출력 코드가 포함된 [bug report](#) 또는 [feature request](#)를 작성하세요. 특정 사용 사례가 [existing issue](#)에서 이미 보고된 경우 공감을 표시하여 지지를 보여주세요.

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 downloads](#)를 참조하세요. 자세한 내용은 [JavaScript ES6/CommonJS 구문](#) 단원을 참조하십시오.

모듈화된 패키지

JavaScript용 SDK(v2) 버전 2에서는 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 몇 가지 AWS 서비스만 사용해야 하는 경우 필요하거나 사용하지 않는 코드로 애플리케이션 크기를 늘려야 합니다.

v3에서는 필요한 개별 AWS 서비스만 로드하고 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 로드하고 사용할 수 있을 뿐만 아니라 필요한 서비스 명령만 로드하고 사용할 수 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";

const dbclient = new DynamoDBClient({ region: "us-west-2" });

try {
  const results = await dbclient.send(new ListTablesCommand());

  for (const item of results.TableNames) {
    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 줄

어듭니다. 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 직접 호출

v3에서 v2 또는 v3 명령을 사용하여 작업을 수행할 수 있습니다. v3 명령을 사용하려면 명령과 필요한 AWS 서비스 패키지 클라이언트를 가져오고 async/await 패턴을 사용하는 .send 메서드를 사용하여 명령을 실행합니다.

v2 명령을 사용하려면 필요한 AWS 서비스 패키지를 가져오고 콜백 또는 비동기/대기 패턴을 사용하여 패키지에서 직접 v2 명령을 실행합니다.

v3 명령 사용

v3는 각 AWS 서비스 패키지에 대한 명령 세트를 제공하여 해당 AWS 서비스에 대한 작업을 수행할 수 있도록 합니다. 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" });
const tableParams = {
  TableName: TABLE_NAME
};
```

```
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 코드와 같이 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 dynamoDB = new DynamoDB({ region: 'us-west-2' });
var tableParams = {
  TableName: TABLE_NAME
};
async function run() => {
  try {
    const data = await dynamoDB.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);
    }
  })
};
run();
```

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 직접 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);
  },
  {
    name: "my-middleware",
    override: true,
    step: "build"
  }
);
```

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

AWS SDK for JavaScript v2와 v3의 차이점은 무엇인가요?

이 섹션에서는 AWS SDK for JavaScript v2에서 v3로의 주목할 만한 변경 사항을 살펴봅니다. v3는 v2를 모듈식으로 재작성한 것이므로 일부 기본 개념은 v2와 v3 간에 다릅니다. [블로그 게시물](#)에서 이러한 변경 사항에 대해 알아볼 수 있습니다. 다음 블로그 게시물을 통해 속도를 높일 수 있습니다.

- [AWS SDK for JavaScript의 모듈식 패키지](#)
- [모듈식 AWS SDK for JavaScript의 미들웨어 스택 소개](#)

AWS SDK for JavaScript v2에서 v3로의 인터페이스 변경 사항 요약은 다음과 같습니다. 목표는 여러분이 이미 익숙한 v2 API의 v3 대응 버전을 쉽게 찾을 수 있도록 돕는 것입니다.

주제

- [클라이언트 생성자](#)
- [보안 인증 공급자](#)
- [Amazon S3 고려 사항](#)
- [DynamoDB 문서 클라이언트](#)
- [웨이터 및 서명자](#)
- [특정 서비스 클라이언트에 대한 참고 사항](#)

클라이언트 생성자

이 목록은 [v2 구성 파라미터](#)로 인덱싱됩니다.

- [computeChecksums](#)
 - v2: 서비스가 페이로드 본문을 수락할 때 페이로드 본문에 대한 MD5 체크섬을 계산할지 여부(현재 S3에서만 지원됨).
 - v3: S3의 해당 명령(PutObject, PutBucketCors 등)은 요청 페이로드의에 대한 MD5 체크섬을 자동으로 계산합니다. 명령의 ChecksumAlgorithm 파라미터에 다른 체크섬 알고리즘을 지정하여 다른 체크섬 알고리즘을 사용할 수도 있습니다. 자세한 내용은 [S3 기능 공지](#)에서 확인할 수 있습니다.
- [convertResponseTypes](#)

- v2: 응답 데이터를 구문 분석할 때 유형이 변환되는지 여부입니다.
- v3: 더 이상 사용되지 않습니다. 이 옵션은 타임스탬프 또는 base64 바이너리와 같은 유형을 JSON 응답에서 변환하지 않으므로 유형 안전이 아닌 것으로 간주됩니다.
- [correctClockSkew](#)
 - v2: 클럭 스큐 수정을 적용할지 여부와 스큐된 클라이언트 클럭으로 인해 실패한 요청을 재시도할지 여부입니다.
 - v3: 더 이상 사용되지 않습니다. SDK는 항상 클럭 스큐 수정을 적용합니다.
- [systemClockOffset](#)
 - v2: 모든 서명 시간에 적용할 밀리초 단위의 오프셋 값입니다.
 - v3: 변경 없음.
- [credentials](#)
 - v2: 요청에 서명할 AWS 자격 증명입니다.
 - v3: 변경 없음. 자격 증명을 반환하는 비동기 함수일 수도 있습니다. 함수가 expiration (Date)를 반환하면 만료 날짜/시간이 가까워지면 함수가 다시 직접 호출됩니다.
[AwsAuthInputConfig 자격 증명의 v3 API 참조](#)를 참조하세요.
- [endpointCacheSize](#)
 - v2: 엔드포인트 검색 작업에서 엔드포인트를 저장하는 글로벌 캐시의 크기입니다.
 - v3: 변경 없음.
- [endpointDiscoveryEnabled](#)
 - v2: 서비스에서 제공하는 엔드포인트를 사용하여 작업을 직접 호출할지 여부입니다.
 - v3: 변경 없음.
- [hostPrefixEnabled](#)
 - v2: 요청 파라미터를 호스트 이름 접두사로 마샬링할지 여부입니다.
 - v3: 더 이상 사용되지 않습니다. SDK는 필요한 경우 항상 호스트 이름 접두사를 주입합니다.
- [httpOptions](#)

하위 수준 HTTP 요청에 전달할 옵션 세트입니다. 이러한 옵션은 v3에서 다르게 집계됩니다. 새 requestHandler를 제공하여 구성할 수 있습니다. 다음은 Node.js 런타임에서 http 옵션을 설정하는 예제입니다. [NodeHttpHandler에 대한 v3 API 참조](#)에서 자세한 내용을 확인할 수 있습니다.

모든 v3 요청은 기본적으로 HTTPS를 사용합니다. 사용자 지정 httpsAgent만 제공하면 됩니다.

```
const { Agent } = require("https");
const { Agent: HttpAgent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");
```

```
const dynamodbClient = new DynamoDBClient({
```

```
requestHandler: new NodeHttpHandler({
  httpsAgent: new Agent({
    /*params*/
  }),
  connectionTimeout: /*number in milliseconds*/,
  socketTimeout: /*number in milliseconds*/
}),
});
```

http를 사용하는 사용자 지정 엔드포인트를 전달하는 경우 httpAgent를 제공해야 합니다.

```
const { Agent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");

const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: new Agent({
      /*params*/
    }),
  }),
  endpoint: "http://example.com",
});
```

클라이언트가 브라우저에서 실행 중인 경우 다른 옵션 세트를 사용할 수 있습니다.

[FetchHttpHandler에 대한 v3 API 참조](#)에서 자세한 내용을 확인할 수 있습니다.

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
const dynamodbClient = new DynamoDBClient({
  requestHandler: new FetchHttpHandler({
    requestTimeout: /* number in milliseconds */
  }),
});
```

httpOptions의 각 옵션은 아래에 지정되어 있습니다.

- proxy
 - v2: 요청을 프록시할 URL입니다.
 - v3: [Node.js용 프록시 구성](#)에 따라 에이전트로 프록시를 설정할 수 있습니다.
- agent
 - v2: HTTP 요청을 수행할 에이전트 객체입니다. 연결 풀링에 사용됩니다.
 - v3: 위의 예제와 같이 httpAgent 또는 httpsAgent를 구성할 수 있습니다.

- `connectTimeout`
 - v2: `connectTimeout`밀리초 후에 서버와의 연결을 설정하지 못한 후 소켓을 제한 시간으로 설정합니다.
 - v3: `connectionTimeout`은 [NodeHttpHandler 옵션에서](#) 사용할 수 있습니다.
- `timeout`
 - v2: 요청이 자동으로 종료되기까지 걸릴 수 있는 밀리초 수입니다.
 - v3: `socketTimeout`은 [NodeHttpHandler 옵션에서](#) 사용할 수 있습니다.
- `xhrAsync`
 - v2: SDK가 비동기 HTTP 요청을 전송할지 여부입니다.
 - v3: 더 이상 사용되지 않습니다. 요청은 항상 비동기식입니다.
- `xhrWithCredentials`
 - v2: XMLHttpRequest 객체의 'withCredentials' 속성을 설정합니다.
 - v3: 사용할 수 없습니다. SDK는 [기본 가져오기 구성](#)을 상속합니다.
- [logger](#)
 - v2: 요청에 대한 정보를 로깅하기 위해 `.write()`(스트림 등) 또는 `.log()`(콘솔 객체 등)에 응답하는 객체입니다.
 - v3: 변경 없음. v3에서는 보다 세분화된 로그를 사용할 수 있습니다.
- [maxRedirects](#)
 - v2: 서비스 요청에 대해 따를 최대 리디렉션 수입니다.
 - v3: 더 이상 사용되지 않습니다. SDK는 의도하지 않은 교차 리전 요청을 방지하기 위해 리디렉션을 따르지 않습니다.
- [maxRetries](#)
 - v2: 서비스 요청에 대해 수행할 최대 재시도 횟수입니다.
 - v3: `maxAttempts`로 변경되었습니다. [RetryInputConfig에 대한 v3 API 참조](#)에서 자세한 내용을 참조하세요. `maxAttempts`는 `maxRetries` + 1이어야 합니다.
- [paramValidation](#)
 - v2: 요청을 보내기 전에 작업 설명과 비교하여 입력 파라미터를 검증해야 하는지 여부입니다.
 - v3: 더 이상 사용되지 않습니다. SDK는 런타임 시 클라이언트 측에서 검증을 수행하지 않습니다.
- [region](#)
 - v2: 서비스 요청을 보낼 리전입니다.
 - v3: 변경 없음. 리전 문자열을 반환하는 비동기 함수일 수도 있습니다.
- [retryDelayOptions](#)
 - v2: 재시도 가능한 오류에 대한 재시도 지연을 구성하는 옵션 세트입니다.

- v3: 더 이상 사용되지 않습니다. SDK는 `retryStrategy` 클라이언트 생성자 옵션을 사용하여 보다 유연한 재시도 전략을 지원합니다. [v3 API 참조](#)에서 자세한 내용을 참조하세요.
- [s3BucketEndpoint](#)
 - v2: 제공된 엔드포인트가 개별 버킷을 처리하는지 여부(루트 API 엔드포인트를 처리하는 경우 `false`).
 - v3: `bucketEndpoint`로 변경되었습니다. [bucketEndpoint에 대한 v3 API 참조](#)에서 자세한 내용을 참조하세요. `true`로 설정하면 Bucket 요청 파라미터에서 요청 엔드포인트를 지정했을 때 원래 엔드포인트를 덮어씁니다. v2에서는 클라이언트 생성자의 요청 엔드포인트가 Bucket 요청 파라미터를 덮어씁니다.
- [s3DisableBodySigning](#)
 - v2: 서명 버전 v4를 사용할 때 S3 본문 서명을 비활성화할지 여부입니다.
 - v3: 이름이 `applyChecksum`으로 바뀌었습니다.
- [s3ForcePathStyle](#)
 - v2: S3 객체에 대한 경로 스타일 URL을 강제 적용할지 여부입니다.
 - v3: 이름이 `forcePathStyle`으로 바뀌었습니다.
- [s3UseArnRegion](#)
 - v2: 요청된 리소스의 ARN에서 추론된 리전으로 요청 리전을 재정의할지 여부입니다.
 - v3: 이름이 `useArnRegion`으로 바뀌었습니다.
- [s3UseEast1RegionalEndpoint](#)
 - v2: 리전이 'us-east-1'로 설정된 경우 글로벌 엔드포인트 또는 'us-east-1' 리전 엔드포인트에 S3 요청을 보낼지 여부입니다.
 - v3: 더 이상 사용되지 않습니다. 리전이 `us-east-1`로 설정된 경우 S3 클라이언트는 항상 리전 엔드포인트를 사용합니다. 리전을 `aws-global`로 설정하여 S3 글로벌 엔드포인트에 요청을 보낼 수 있습니다.
- [signatureCache](#)
 - v2: 서명 요청에 사용할 서명(API 구성을 재정의함)이 캐시되는지 여부입니다.
 - v3: 더 이상 사용되지 않습니다. SDK는 해시된 서명 키를 항상 캐시합니다.
- [signatureVersion](#)
 - v2: 요청에 서명할 시그니처 버전입니다(API 구성을 재정의함).
 - v3: 더 이상 사용되지 않습니다. v2 SDK에서 지원되는 서명 V2는에서 더 이상 사용되지 않습니다 AWS. v3는 서명 v4만 지원합니다.
- [sslEnabled](#)
 - v2: 요청에 SSL을 활성화할지 여부입니다.
 - v3: 이름이 `tls`으로 바뀌었습니다.
- [stsRegionalEndpoints](#)

- v2: 글로벌 엔드포인트 또는 리전 엔드포인트에 STS 요청을 보낼지 여부입니다.
- v3: 더 이상 사용되지 않습니다. STS 클라이언트는 특정 리전으로 설정된 경우 항상 리전 엔드포인트를 사용합니다. 리전을 `aws-global`로 설정하여 STS 글로벌 엔드포인트에 요청을 보낼 수 있습니다.
- [useAccelerateEndpoint](#)
 - v2: Accelerate 엔드포인트를 S3 서비스와 함께 사용할지 여부입니다.
 - v3: 변경 없음.

보안 인증 공급자

v2에서 JavaScript용 SDK는 선택할 수 있는 자격 증명 공급자 목록과 Node.js에서 기본적으로 사용 가능한 자격 증명 공급자 체인을 제공하며, 이를 통해 가장 일반적인 공급자의 AWS 자격 증명을 로드하려고 합니다. SDK for JavaScript v3는 자격 증명 공급자의 인터페이스를 간소화하므로 사용자 지정 자격 증명 공급자를 더 쉽게 사용하고 쓸 수 있습니다. 새로운 자격 증명 공급자 체인 외에도 SDK for JavaScript v3는 모두 v2와 동등한 것을 제공하는 것을 목표로 하는 자격 증명 공급자 목록을 제공합니다.

다음은 v2의 모든 자격 증명 공급자와 v3의 해당 자격 증명 공급자입니다.

기본 자격 증명 공급자

기본 자격 증명 공급자는 명시적으로 제공하지 않는 경우 SDK for JavaScript가 AWS 자격 증명을 확인하는 방법입니다.

- v2: Node.js의 [CredentialProviderChain](#)은 소스의 자격 증명을 다음 순서로 확인합니다.
 - [환경 변수](#)
 - [공유 자격 증명 파일](#)
 - [ECS 컨테이너 자격 증명](#)
 - [외부 프로세스 생성](#)
 - [지정된 파일의 OIDC 토큰](#)
 - [Amazon EC2 인스턴스 메타데이터입니다.](#)

위의 자격 증명 공급자 중 하나가 AWS 자격 증명을 확인하지 못하면 유효한 자격 증명이 확인될 때까지 체인이 다음 공급자로 돌아가고 모든 공급자가 실패하면 체인에서 오류가 발생합니다.

브라우저 및 React Native 런타임에서 자격 증명 체인은 비어 있으며 자격 증명을 명시적으로 설정해야 합니다.

- v3: [defaultProvider](#). 자격 증명 소스와 대체 순서는 v3에서 변경되지 않습니다. 또한 [AWS IAM Identity Center 자격 증명](#)을 지원합니다.

임시 자격 증명

- v2: [ChainableTemporaryCredentials](#)는 AWS.STS에서 검색된 임시 자격 증명을 나타냅니다. 추가 파라미터가 없으면 AWS.STS.getSessionToken() 작업에서 자격 증명을 가져옵니다. IAM 역할이 제공된 경우, AWS.STS.assumeRole() 작업이 해당 역할의 자격 증명을 가져오는 데 사용됩니다. AWS.ChainableTemporaryCredentials는 AWS.TemporaryCredentials와는 다른 방식으로 masterCredentials 및 갱신을 처리합니다. AWS.ChainableTemporaryCredentials는 사용자가 전달한 masterCredentials를 사용하여 만료된 자격 증명을 갱신하여 STS 자격 증명의 체인 연결을 지원합니다. 그러나 AWS.TemporaryCredentials는 인스턴스화 중에 masterCredentials를 반복적으로 축소하므로 중간 임시 자격 증명에 필요한 자격 증명을 새로 고칠 수 없습니다.

원본 [TemporaryCredentials](#)는 v2의 ChainableTemporaryCredentials에서 더 이상 사용되지 않습니다.

- v3: [fromTemporaryCredentials](#). @aws-sdk/credential-providers 패키지에서 fromTemporaryCredentials()를 호출할 수 있습니다. 다음은 그 예입니다.

```
import { FooClient } from "@aws-sdk/client-foo";
import { fromTemporaryCredentials } from "@aws-sdk/credential-providers"; // ES6
import
// const { FooClient } = require("@aws-sdk/client-foo");
// const { fromTemporaryCredentials } = require("@aws-sdk/credential-providers"); //
CommonJS import

const sourceCredentials = {
  // A credential can be a credential object or an async function that returns a
  credential object
};
const client = new FooClient({
  credentials: fromTemporaryCredentials({
    masterCredentials: sourceCredentials,
    params: { RoleArn },
  }),
});
```

Amazon Cognito Identity 자격 증명

일반적으로 브라우저에서 사용되는 Amazon Cognito Identity 서비스에서 자격 증명을 로드합니다.

- v2: [CognitoIdentityCredentials](#) Amazon Cognito Identity 서비스를 사용하여 STS 웹 ID 페더레이션에서 검색된 자격 증명을 나타냅니다.
- v3: [Cognito Identity Credential Provider @aws/credential-providers 패키지](#)는 두 개의 자격 증명 공급자 함수를 제공합니다. 하나는 [fromCognitoIdentity](#)로, 자격 증명 ID를 받아 `cognitoIdentity:GetCredentialsForIdentity`를 직접 호출합니다. 다른 하나는 [fromCognitoIdentityPool](#)로, 자격 증명 풀 ID를 받아 첫 번째 간접 호출 시 `cognitoIdentity:GetId`를 직접 호출한 후 직접 호출합니다 `fromCognitoIdentity`. 후자의 후속 간접 호출은 `GetId`를 다시 간접 호출하지 않습니다.

공급자는 [Amazon Cognito 개발자 안내서](#)에 설명된 ‘간소화된 흐름’을 구현합니다. 직접 호출 `cognito:GetOpenIdToken` 및 `sts:AssumeRoleWithWebIdentity`를 포함하는 ‘클래식 흐름’은 지원되지 않습니다. 필요한 경우 [기능 요청](#)을 시작해 주세요.

```
// fromCognitoIdentityPool example
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers"; // ES6
import
// const { fromCognitoIdentityPool } = require("@aws-sdk/credential-providers"); //
CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

```
// fromCognitoIdentity example
import { fromCognitoIdentity } from "@aws-sdk/credential-providers"; // ES6 import
```

```
// const { fromCognitoIdentity } = require("@aws-sdk/credential-provider-cognito-identity"); // CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentity({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityId: "us-east-1:128d0a74-c82f-4553-916d-90053e4a8b0f",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

Amazon EC2 메타데이터(IMDS) 자격 증명

Amazon EC2 인스턴스의 메타데이터 서비스에서 받은 자격 증명을 나타냅니다.

- v2: [EC2MetadataCredentials](#)
- v3: [fromInstanceMetadata](#). Amazon EC2 인스턴스 메타데이터 서비스에서 자격 증명을 소싱할 자격 증명 공급자를 생성합니다.

```
import { fromInstanceMetadata } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromInstanceMetadata } = require("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromInstanceMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```


Amazon ECS 자격 증명

지정된 URL에서 받은 자격 증명을 나타냅니다. 이 공급자는 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` 또는 `AWS_CONTAINER_CREDENTIALS_FULL_URI` 환경 변수로 지정된 URI에서 임시 자격 증명을 요청합니다.

- v2: `ECSCredentials` 또는 [RemoteCredentials](#)
- v3: [fromContainerMetadata](#). Amazon ECS 컨테이너 메타데이터 서비스에서 자격 증명을 소싱할 자격 증명 공급자를 생성합니다.

```
import { fromContainerMetadata } from "@aws-sdk/credential-providers"; // ES6 import

const client = new FooClient({
  credentials: fromContainerMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

파일 시스템 자격 증명

- v2: [FileSystemCredentials](#). 디스크에 있는 JSON 파일의 자격 증명을 나타냅니다.
- v3: 더 이상 사용되지 않습니다. JSON 파일을 명시적으로 읽고 클라이언트에 제공할 수 있습니다. 필요한 경우 [기능 요청](#)을 시작해 주세요.

SAML 자격 증명 공급자

- v2: [SAMLCredentials](#) STS SAML 지원에서 검색된 자격 증명을 나타냅니다.
- v3: 사용할 수 없습니다. 필요한 경우 [기능 요청](#)을 시작해 주세요.

공유 자격 증명 파일 자격 증명

공유 자격 증명 파일에서 자격 증명을 로드합니다(기본값은 `~/.aws/credentials` 또는 `AWS_SHARED_CREDENTIALS_FILE` 환경 변수로 정의됨). 이 파일은 다양한 AWS SDK 및 도구에서 지원됩니다. 자세한 내용은 [공유 구성 및 자격 증명 파일 문서](#)를 참조하세요.

- v2: [SharedIniFileCredentials](#)

- v3: [fromIni](#)

```
import { fromIni } from "@aws-sdk/credential-providers";
// const { fromIni } from("@aws-sdk/credential-providers");

const client = new FooClient({
  credentials: fromIni({
    configFilepath: "~/.aws/config", // Optional
    filepath: "~/.aws/credentials", // Optional
    mfaCodeProvider: async (mfaSerial) => {
      // implement a pop-up asking for MFA code
      return "some_code";
    }, // Optional
    profile: "default", // Optional
    clientConfig: { region }, // Optional
  }),
});
```

웹 ID 자격 증명

디스크의 파일에서 OIDC 토큰을 사용하여 자격 증명을 검색합니다. Amazon EKS에서 일반적으로 사용됩니다.

- v2: [TokenFileWebIdentityCredentials](#)
- v3: [fromTokenFile](#)

```
import { fromTokenFile } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromTokenFile } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromTokenFile({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

웹 ID 페더레이션 자격 증명

STS 웹 ID 페더레이션 지원에서 자격 증명을 검색합니다.

- v2: [WebIdentityCredentials](#)
- v3: [fromWebToken](#)

```
import { fromWebToken } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromWebToken } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromWebToken({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Amazon S3 고려 사항

Amazon S3 멀티파트 업로드

v2에서 Amazon S3 클라이언트에는 [Amazon S3에서 제공하는 멀티파트 업로드](#) 기능을 사용하여 대용량 객체 업로드를 지원하는 [upload\(\)](#) 작업이 포함되어 있습니다.

v3에서는 [@aws-sdk/lib-storage](#) 패키지를 사용할 수 있습니다. v2 `upload()` 작업에서 제공되는 모든 기능을 지원하며 Node.js 및 브라우저 런타임을 모두 지원합니다.

Amazon S3에 미리 서명된 URL

v2에서 Amazon S3 클라이언트에는 사용자가 Amazon S3에서 객체를 업로드하거나 다운로드하는 데 사용할 수 있는 URL을 생성하는 [getSignedUrl\(\)](#) 및 [getSignedUrlPromise\(\)](#) 작업이 포함되어 있습니다.

v3에서는 [@aws-sdk/s3-request-presigner](#) 패키지를 사용할 수 있습니다. 이 패키지에는 `getSignedUrl()` 및 `getSignedUrlPromise()` 작업 모두에 대한 함수가 포함되어 있습니다. 이 [블로그 게시물](#)에서 이 패키지의 세부 정보를 설명합니다.

Amazon S3 리전 리디렉션

잘못된 리전이 Amazon S3 클라이언트에 전달되고 후속 `PermanentRedirect`(상태 301) 오류가 발생하는 경우 v3의 Amazon S3 클라이언트는 리전 리디렉션(이전에는 v2의 Amazon S3 Global Client라고 함)을 지원합니다. 클라이언트 구성에서 [followRegionRedirects](#) 플래그를 사용하여 Amazon S3 클라이언트가 리전 리디렉션을 따르고 글로벌 클라이언트로서 해당 기능을 지원하도록 할 수 있습니다.

Note

상태가 301인 `PermanentRedirect` 오류를 수신하면 실패한 요청이 수정된 리전으로 재시도 되므로 이 기능으로 인해 추가 지연 시간이 발생할 수 있습니다. 이 기능은 버킷의 리전을 사전에 알지 못하는 경우에만 사용해야 합니다.

Amazon S3 스트리밍 및 버퍼링된 응답

v3 SDK는 잠재적으로 큰 응답을 버퍼링하지 않는 것을 선호합니다. 이는 일반적으로 v2에서 `Buffer`를 반환했지만 v3에서 `Stream`을 반환하는 Amazon S3 `GetObject` 작업에서 발생합니다.

Node.js의 경우 스트림 또는 가비지 수집 클라이언트 또는 요청 핸들러를 사용하여 소켓을 해제하여 새 트래픽에 대한 연결을 열어 두어야 합니다.

```
// v2
const get = await s3.getObject({ ... }).promise(); // this buffers consumes the stream already.
```

```
// v3, consume the stream to free the socket
const get = await s3.getObject({ ... }); // object .Body has unconsumed stream
const str = await get.Body.transformToString(); // consumes the stream
```

```
// other ways to consume the stream include writing it to a file,
// passing it to another consumer like an upload, or buffering to
// a string or byte array.
```

자세한 내용은 [소켓 소진](#) 섹션을 참조하세요.

DynamoDB 문서 클라이언트

v3에서 DynamoDB 문서 클라이언트의 기본 사용

- v2에서는 [AWS.DynamoDB.DocumentClient](#) 클래스를 사용하여 배열, 숫자 및 객체와 같은 기본 JavaScript 유형을 사용하여 DynamoDB API를 직접 호출할 수 있습니다. 이렇게 하면 속성 값의 개념을 추상화하여 Amazon DynamoDB에서 항목 작업을 간소화할 수 있습니다.
- v3에서는 동등한 [@aws-sdk/lib-dynamodb](#) 클라이언트를 사용할 수 있습니다. v3 SDK의 일반 서비스 클라이언트와 유사하지만 생성자에서 기본 DynamoDB 클라이언트를 사용한다는 차이점이 있습니다.

예제:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb"; // ES6 import
// const { DynamoDBClient } = require("@aws-sdk/client-dynamodb"); // CommonJS import
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb"; // ES6 import
// const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb"); // CommonJS import

// Bare-bones DynamoDB Client
const client = new DynamoDBClient({});

// Bare-bones document client
const ddbDocClient = DynamoDBDocumentClient.from(client); // client is DynamoDB client

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "1",
      content: "content from DynamoDBDocumentClient",
    },
  })
);
```

마샬링 시의 **Undefined** 값

- v2에서는 DynamoDB에 대한 마샬링 프로세스 중에 객체의 undefined 값이 자동으로 생략되었습니다.

- v3에서는 @aws-sdk/lib-dynamodb의 기본 마샬링 동작이 변경되었습니다. 값이 undefined인 객체는 더 이상 생략되지 않습니다. v2의 기능에 맞게 개발자는 DynamoDB 문서 클라이언트의 marshallOptions에서 removeUndefinedValues를 true로 명시적으로 설정해야 합니다.

예제:

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

const client = new DynamoDBClient({});

// The DynamoDBDocumentClient is configured to handle undefined values properly
const ddbDocClient = DynamoDBDocumentClient.from(client, {
  marshallOptions: {
    removeUndefinedValues: true
  }
});

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "123",
      content: undefined // This value will be automatically omitted.
      array: [1, undefined], // The undefined value will be automatically omitted.
      map: { key: undefined }, // The "key" will be automatically omitted.
      set: new Set([1, undefined]), // The undefined value will be automatically
      omitted.
    }
  })
);
```

[패키지 README](#)에서 더 많은 예제와 구성을 사용할 수 있습니다.

웨이터 및 서명자

이 페이지에서는 AWS SDK for JavaScript v3에서 웨이터 및 서명자를 사용하는 방법을 설명합니다.

Waiters

v2에서는 모든 웨이터가 서비스 클라이언트 클래스에 바인딩되며 클라이언트가 대기할 설계된 상태를 웨이터의 입력에 지정해야 합니다. 예를 들어 새로 생성된 버킷이 준비될 때까지 기다리려면 [waitFor\("bucketExists"\)](#)를 호출해야 합니다.

v3에서는 애플리케이션에 웨이터가 필요하지 않은 경우 웨이터를 가져올 필요가 없습니다. 또한 원하는 특정 상태를 기다리는 데 필요한 웨이터만 가져올 수 있습니다. 따라서 번들 크기를 줄이고 성능을 개선할 수 있습니다. 다음은 생성 후 버킷이 준비될 때까지 기다리는 예입니다.

```
import { S3Client, CreateBucketCommand, waitUntilBucketExists } from "@aws-sdk/client-s3"; // ES6 import
// const { S3Client, CreateBucketCommand, waitUntilBucketExists } = require("@aws-sdk/client-s3"); // CommonJS import

const Bucket = "BUCKET_NAME";
const client = new S3Client({ region: "REGION" });
const command = new CreateBucketCommand({ Bucket });

await client.send(command);
await waitUntilBucketExists({ client, maxWaitTime: 60 }, { Bucket });
```

[AWS SDK for JavaScript v3의 웨이터 블로그 게시물](#)에서 웨이터를 구성하는 방법을 모두 찾을 수 있습니다.

Amazon CloudFront 서명자

v2에서는 [AWS.CloudFront.Signer](#)를 사용하여 제한된 Amazon CloudFront 배포에 액세스하기 위한 요청에 서명할 수 있습니다.

v3에는 [@aws-sdk/cloudfront-signer](#) 패키지에 제공된 것과 동일한 유틸리티가 있습니다.

Amazon RDS 서명자

v2에서는 [AWS.RDS.Signer](#)를 사용하여 Amazon RDS 데이터베이스에 대한 인증 토큰을 생성할 수 있습니다.

v3에서는 [@aws-sdk/rds-signer](#) 패키지에서 유사한 유틸리티 클래스를 사용할 수 있습니다.

Amazon Polly 서명자

v2에서는 [AWS.Polly.Presigner](#)를 사용하여 Amazon Polly 서비스에서 합성한 음성에 대한 서명된 URL을 생성할 수 있습니다.

v3에서는 [@aws-sdk/polly-request-presigner](#) 패키지에서 유사한 유틸리티 함수를 사용할 수 있습니다.

특정 서비스 클라이언트에 대한 참고 사항

AWS Lambda

Lambda 간접 호출 응답 유형은 v2와 v3에서 서로 다릅니다.

```
// v2
import { Lambda } from "@aws-sdk/client-lambda";
import AWS from "aws-sdk";

const lambda = new AWS.Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
}).promise();

// in v2, Lambda::invoke::Payload is automatically converted to string via a
// specific code customization.
const payloadIsString = typeof invoke.Payload === "string";
console.log("Invoke response payload type is string:", payloadIsString);

const payloadObject = JSON.parse(invoke.Payload);
console.log("Invoke response object", payloadObject);
```

```
// v3
const lambda = new Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
});

// in v3, Lambda::invoke::Payload is not automatically converted to a string.
// This is to reduce the number of customizations that create inconsistent behaviors.
const payloadIsByteArray = invoke.Payload instanceof Uint8Array;
```



```
console.log("Invoke response payload type is Uint8Array:", payloadIsByteArray);

// To maintain the old functionality, only one additional method call is needed:
// v3 adds a method to the Uint8Array called transformToString.
const payloadObject = JSON.parse(invoke.Payload.transformToString());
console.log("Invoke response object", payloadObject);
```

Amazon SQS

MD5 체크섬

메시지 본문의 MD5 체크섬 계산을 건너뛰려면 구성 객체에서 `md5`를 `false`로 설정합니다. 그렇지 않으면 SDK는 기본적으로 메시지 전송을 위한 체크섬을 계산하고 검색된 메시지에 대한 체크섬을 검증합니다.

```
// Example: Skip MD5 checksum in Amazon SQS
import { SQS } from "@aws-sdk/client-sqs";

new SQS({
  md5: false // note: only available in v3.547.0 and higher
});
```

이를 입력 파라미터로 사용하는 Amazon SQS 작업에서 사용자 지정 `QueueUrl`을 사용할 때 v2에서는 Amazon SQS 클라이언트의 기본 엔드포인트를 재정의하는 사용자 지정 `QueueUrl`을 제공할 수 있었습니다.

다중 리전 메시지

v3에서는 리전당 하나의 클라이언트를 사용해야 합니다. AWS 리전은 클라이언트 수준에서 초기화되며 요청 간에 변경되지 않습니다.

```
import { SQS } from "@aws-sdk/client-sqs";

const sqsClients = {
  "us-east-1": new SQS({ region: "us-east-1" }),
  "us-west-2": new SQS({ region: "us-west-2" }),
};

const queues = [
  { region: "us-east-1", url: "https://sqs.us-east-1.amazonaws.com/{AWS_ACCOUNT}/MyQueue" },
```

```

    { region: "us-west-2", url: "https://sqs.us-west-2.amazonaws.com/{AWS_ACCOUNT}/
    MyOtherQueue" },
  ];

  for (const { region, url } of queues) {
    const params = {
      MessageBody: "Hello",
      QueueUrl: url,
    };
    await sqsClients[region].sendMessage(params);
  }

```

사용자 지정 엔드포인트

v3에서 사용자 지정 엔드포인트, 즉 기본 퍼블릭 Amazon SQS 엔드포인트와 다른 엔드포인트를 사용하는 경우 항상 Amazon SQS 클라이언트와 `QueueUrl` 필드에 엔드포인트를 설정해야 합니다.

```

import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  // client endpoint should be specified in v3 when not the default public SQS endpoint
  // for your region.
  // This is required for versions <= v3.506.0
  // This is optional but recommended for versions >= v3.507.0 (a warning will be
  // emitted)
  endpoint: "https://my-custom-endpoint:8000/",
});

await sqs.sendMessage({
  QueueUrl: "https://my-custom-endpoint:8000/1234567/MyQueue",
  Message: "hello",
});

```

사용자 지정 엔드포인트를 사용하지 않는 경우 클라이언트에서 `endpoint`를 설정할 필요가 없습니다.

```

import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  region: "us-west-2",
});

await sqs.sendMessage({
  QueueUrl: "https://sqs.us-west-2.amazonaws.com/1234567/MyQueue",
});

```

```
Message: "hello",
});
```

보충 설명서

다음 표에는 AWS SDK for JavaScript(v3)를 사용하고 이해하는 데 도움이 되는 보충 설명서 링크가 포함되어 있습니다.

명칭	Notes
SDK 클라이언트	SDK 클라이언트 및 구성 가능한 일반적인 생성자 파라미터 초기화에 대한 정보입니다.
업그레이드 참고 사항(2.x에서 3.x로)	AWS SDK for JavaScript(v2)에서 업그레이드하는 방법에 대한 정보입니다.
AWS Lambda Node.js 런타임에서 AWS SDK for JavaScript(v3) 사용	AWS SDK for JavaScript(v3)를 사용하여 AWS Lambda 내에서 작업하는 모범 사례입니다.
성능.	AWS SDK 팀이 SDK의 성능을 최적화한 방법에 대한 정보와 SDK를 효율적으로 실행하도록 구성하기 위한 팁이 포함되어 있습니다.
TypeScript	AWS SDK for JavaScript(v3)와 관련된 TypeScript 팁 및 FAQ.
오류 처리	AWS SDK for JavaScript(v3)와 관련된 오류를 처리하기 위한 팁입니다.
유효 사례	AWS SDK for JavaScript(v3) 사용에 대한 일반 권장 사항입니다.

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 feed](#)를 구독하세요.

변경 사항	설명	날짜
이제 모든 리전의 모든 AWS 서비스 API 엔드포인트에서 TLS 1.3이 지원됩니다.	지원되는 TLS 버전과 TLS 버전 로깅 방법이 업데이트되었습니다.	2025년 4월 10일
@smithy/types를 사용하여 클라이언트 생성	@smithy/types 패키지를 사용하여 클라이언트를 생성하도록 콘텐츠가 업데이트되었습니다.	2025년 2월 15일
체크섬을 통한 데이터 무결성 보호	자동 체크섬 계산에 대한 세부 정보로 콘텐츠가 업데이트되었습니다.	2025년 1월 15일
아마존 S3 체크섬	Amazon S3에서 유연한 체크섬을 사용하는 방법에 대한 섹션이 추가되었습니다.	2025년 1월 1일
DynamoDB에서 계정 기반 엔드포인트 지원	DynamoDB의 계정 기반 엔드포인트에 대한 지원이 AWS SDK for JavaScript 추가되었습니다.	2024년 9월 26일
SDK 로깅에 대한 새로운 주제	요청 로깅을 위한 미들웨어 사용에 대한 정보와 함께 SDK for JavaScript를 사용한 API 직접 호출을 로깅하는 방법을 설명하는 주제가 추가되었습니다.	2024년 9월 26일

관련 공지 사항	Internet Explorer 11에 대한 지원 종료 알림이 포함된 상단 배너가 업데이트되었습니다.	2022년 9월 23일
마이너 업데이트	끊어진 링크 해결 및 명확성을 위한 마이너 업데이트. AWS SDKs 및 도구 참조 가이드에 대한 인식 링크가 추가되었습니다.	2022년 8월 22일
최소 TLS 버전 적용	TLS 1.3에 관한 정보가 추가되었습니다.	2022년 3월 31일
Node.js의 보안 인증 설정 항목 업데이트	Node.js for AWS SDK for JavaScript V3에서 자격 증명 설정에 대한 주제를 업데이트합니다.	2020년 10월 20일
v3로 마이그레이션	AWS SDK for JavaScript v3로 마이그레이션하는 방법을 설명하는 주제가 추가되었습니다.	2020년 10월 20일
시작하기	브라우저에서 시작하기 및 AWS SDK for JavaScript V3용 Node.js 시작하기에 대한 주제가 업데이트되었습니다.	2020년 10월 20일
브라우저 빌더	AWS Browser Builder에 대한 정보는 AWS SDK for JavaScript V3에 필요하지 않으므로 제거되었습니다.	2020년 10월 20일
Amazon Transcribe 서비스 예 업데이트	AWS SDK for JavaScript V3에 대한 Amazon Transcribe 서비스 예제를 업데이트했습니다.	2020년 10월 20일

Amazon Simple Notification Service 서비스 예 업데이트	AWS SDK for JavaScript V3에 대한 Amazon Simple Notification Service 서비스 예제를 업데이트했습니다.	2020년 10월 20일
Amazon Simple Email Service 서비스 예 업데이트	AWS SDK for JavaScript V3에 대한 Amazon Simple Email Service 서비스 예제를 업데이트했습니다.	2020년 10월 20일
Amazon Redshift 서비스 예 업데이트	AWS SDK for JavaScript V3에 대한 Amazon Redshift 서비스 예제를 업데이트했습니다.	2020년 10월 20일
Amazon Lex 서비스 예 업데이트	AWS SDK for JavaScript V3에 대한 Amazon Lex 서비스 예제를 업데이트했습니다.	2020년 10월 20일
AWS Elemental MediaConvert 업데이트된 서비스 예제	AWS SDK for JavaScript V3에 대한 AWS Elemental MediaConvert 서비스 예제를 업데이트했습니다.	2020년 10월 20일
AWS Lambda 업데이트된 서비스 예제	AWS SDK for JavaScript V3에 대한 AWS Lambda 서비스 예제를 업데이트했습니다.	2020년 10월 20일
AWS SDK for JavaScript V3 개발자 안내서 미리 보기	AWS SDK for JavaScript V3 개발자 안내서의 시험판 버전을 릴리스했습니다.	2020년 10월 19일