

개발자 가이드

# AWS SDK for Rust



# AWS SDK for Rust: 개발자 가이드

Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께 사용되어서는 안되며, 고객에게 혼동을 일으키거나 Amazon 브랜드 이미지를 떨어뜨리고 폄하하는 방식으로 이용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

# Table of Contents

란 무엇입니까 AWS SDK for Rust? .....	1
SDK 시작하기 .....	1
SDK 메이저 버전에 대한 유지 관리 및 지원 .....	1
추가 리소스 .....	1
시작 .....	3
를 사용한 SDK 인증 AWS .....	3
AWS 액세스 포털 세션 시작 .....	4
세부 인증 정보 .....	5
입문용 자습서 .....	5
사전 조건 .....	5
첫 번째 SDK 앱 생성 .....	6
기본 사항 .....	7
사전 조건 .....	5
Rust 기본 사항 .....	8
AWS SDK for Rust crate 기본 사항 .....	9
로 작업하기 위한 프로젝트 구성 AWS 서비스 .....	9
Tokio 런타임 .....	10
구성 .....	11
서비스 클라이언트 생성 .....	11
환경에서 클라이언트 구성 .....	12
서비스별 설정에 빌더 패턴 사용 .....	13
고급 명시적 클라이언트 구성 .....	14
동작 버전 .....	14
에서 동작 버전 설정 Cargo.toml .....	15
코드에서 동작 버전 설정 .....	15
보안 인증 공급자 .....	15
자격 증명 공급자 체인 .....	16
명시적 자격 증명 공급자 .....	18
자격 증명 캐싱 .....	18
AWS 리전 선택 .....	18
AWS 리전 공급자 체인 .....	19
구성 옵션 .....	19
클라이언트 엔드포인트 .....	20
사용자 지정 구성 .....	21

예시 .....	24
HTTP .....	25
하이퍼 1.x .....	25
인터셉터 .....	27
인터셉터 등록 .....	29
관찰성 .....	29
로깅 .....	30
작업 구성 재정의 .....	33
재시도 .....	34
기본 재시도 구성 .....	35
최대 시도 횟수 .....	35
지연 및 백오프 .....	36
적응형 재시도 모드 .....	36
시간 초과 .....	37
API 제한 시간 .....	37
스톨 스트림 보호 .....	39
SDK 사용 .....	40
요청을 생성 .....	40
모범 사례 .....	41
가능한 경우 SDK 클라이언트 재사용 .....	41
API 타임아웃 설정 .....	41
동시성 .....	42
용어 .....	42
간단한 예 .....	42
소유권 및 변경 가능성 .....	44
추가 용어! .....	44
더 효율적으로 예제를 다시 작성(단일 스레드 동시성) .....	45
더 효율적으로 예제를 다시 작성(다중 스레드 동시성) .....	47
다중 스레드 앱 디버깅 .....	49
오류 처리 .....	49
서비스 오류 .....	49
오류 메타데이터 .....	50
를 사용하여 세부 오류 인쇄 DisplayErrorContext .....	51
Lambda 함수 생성 .....	53
페이지 매김 .....	53
미리 서명된 URLs 생성 .....	55

기본 사항 사전 지정 .....	55
POST 및 PUT 요청 사전 서명 .....	56
독립 실행형 서명자 .....	57
유닛 테스트 .....	58
Mockall을 사용하여 모의 자동 생성 .....	58
정적 재생 .....	63
Waiters .....	67
코드 예제 .....	69
API Gateway .....	70
작업 .....	71
시나리오 .....	72
AWS 커뮤니티 기여 .....	73
API Gateway Management API .....	73
작업 .....	71
Application Auto Scaling .....	75
작업 .....	71
Aurora .....	76
기본 사항 .....	78
작업 .....	71
Auto Scaling .....	223
기본 사항 .....	78
작업 .....	71
Amazon Bedrock 런타임 .....	257
Anthropic Claude .....	258
Amazon Cognito 자격 증명 공급자 .....	273
작업 .....	71
Amazon Cognito Sync .....	274
작업 .....	71
Firehose .....	276
작업 .....	71
Amazon DocumentDB .....	277
서버리스 예제 .....	278
DynamoDB .....	279
작업 .....	71
시나리오 .....	72
서버리스 예제 .....	278

AWS 커뮤니티 기여 .....	73
Amazon EBS .....	297
작업 .....	71
Amazon EC2 .....	300
기본 사항 .....	78
작업 .....	71
Amazon ECR .....	361
작업 .....	71
Amazon ECS .....	364
작업 .....	71
Amazon EKS .....	366
작업 .....	71
AWS Glue .....	368
기본 사항 .....	78
작업 .....	71
IAM .....	383
기본 사항 .....	78
작업 .....	71
AWS IoT .....	410
작업 .....	71
Kinesis .....	412
작업 .....	71
서버리스 예제 .....	278
AWS KMS .....	420
작업 .....	71
Lambda .....	429
기본 사항 .....	78
작업 .....	71
시나리오 .....	72
서버리스 예제 .....	278
AWS 커뮤니티 기여 .....	73
MediaLive .....	479
작업 .....	71
MediaPackage .....	480
작업 .....	71
Amazon MSK .....	483

서버리스 예제 .....	278
Amazon Polly .....	485
작업 .....	71
시나리오 .....	72
QLDB .....	490
작업 .....	71
Amazon RDS .....	492
서버리스 예제 .....	278
Amazon RDS .....	495
작업 .....	71
Amazon Rekognition .....	496
시나리오 .....	72
Route 53 .....	499
작업 .....	71
Amazon S3 .....	500
기본 사항 .....	78
작업 .....	71
시나리오 .....	72
서버리스 예제 .....	278
SageMaker AI .....	550
작업 .....	71
Secrets Manager .....	552
작업 .....	71
Amazon SES API v2 .....	553
작업 .....	71
시나리오 .....	72
Amazon SNS .....	570
작업 .....	71
시나리오 .....	72
서버리스 예제 .....	278
Amazon SQS .....	576
작업 .....	71
서버리스 예제 .....	278
AWS STS .....	581
작업 .....	71
Systems Manager .....	582

---

작업 .....	71
Amazon Transcribe .....	585
시나리오 .....	72
보안 .....	587
데이터 보호 .....	587
규정 준수 검증 .....	588
인프라 보안 .....	589
최소 TLS 버전 적용 .....	590
SDK에서 사용하는 상자 .....	592
Smithy 상자 .....	592
SDK와 함께 사용되는 상자 .....	592
기타 상자 .....	593
문서 기록 .....	594
.....	dxcv



# 란 무엇입니까 AWS SDK for Rust?

Rust는 안전, 속도 및 동시성이라는 세 가지 목표에 초점을 맞춘 가비지 수집기가 없는 시스템 프로그래밍 언어입니다.

AWS SDK for Rust (SDK)는 Amazon Web Services 인프라 서비스와 상호 작용할 수 있는 Rust APIs를 제공합니다. SDK를 사용하면 Amazon S3, Amazon EC2, DynamoDB 등을 기반으로 애플리케이션을 구축할 수 있습니다.

## 주제

- [SDK 시작하기](#)
- [SDK 메이저 버전에 대한 유지 관리 및 지원](#)
- [추가 리소스](#)

## SDK 시작하기

SDK를 처음 사용하는 경우 먼저 읽는 것이 좋습니다 [시작하기 AWS SDK for Rust](#).

에 요청하기 위해 서비스 클라이언트를 생성하고 구성하는 방법을 포함한 구성 및 설정은 섹션을 [AWS 서비스 참조하세요 구성 AWS SDK for Rust](#).

SDK 사용에 대한 자세한 내용은 섹션을 참조하세요 [사용 AWS SDK for Rust](#).

Rust 코드 예제의 전체 목록은 섹션을 참조하세요 [코드 예제](#).

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

## 추가 리소스

이 가이드 외에도 SDK 개발자에게 유용한 온라인 리소스는 다음과 같습니다.

- [AWS SDKs 및 도구 참조 가이드](#): AWS SDKs 포함합니다.
- [Rust Programming Language 웹 사이트](#)
- [AWS SDK for Rust API Reference](#)
- [AWS 개발자 도구 블로그](#)
- GitHub의 [AWS SDK for Rust 소스 코드](#)
- [AWS 코드 샘플 카탈로그](#)

# 시작하기 AWS SDK for Rust

SDK를 설치, 설정 및 사용하여 프로그래밍 방식으로 AWS 리소스에 액세스하는 Rust 애플리케이션을 생성하는 방법을 알아봅니다.

주제

- [를 사용한 SDK 인증 AWS](#)
- [에 대한 Hello 자습서 AWS SDK for Rust](#)
- [기본 사항](#)

## 를 사용한 SDK 인증 AWS

를 사용하여 개발할 AWS 때 코드가 인증하는 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 인증 방법을 받지 않은 신규 사용자를 설정하는 것이 좋습니다 AWS IAM Identity Center. 이 방법에는 구성이 용이하고 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 .
- 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 SDK가 AWS 요청에 사용하는 기본 값 AWS 리전 이 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
```

SDK for Rust는 IAM Identity Center 인증을 사용하기 위해 애플리케이션에 추가 패키지(예: SSO 및 SS00IDC)를 추가할 필요가 없습니다.

## 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.

## 세부 인증 정보

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

에 액세스할 때 임시 자격 증명을 사용합니다 AWS. 인적 사용자에게 대한 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](#).
- SDK for Rust에서 지원하는 다른 자격 증명 공급자에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [표준화된 자격 증명 공급자](#)를 참조하세요. AWS SDKs

## 에 대한 Hello 자습서 AWS SDK for Rust

### 사전 조건

를 사용하려면 Rust 및 Cargo가 설치되어 있어야 AWS SDK for Rust합니다.

- Rust 도구 체인 설치: <https://www.rust-lang.org/tools/install>
- 명령을 실행하여 cargo-component [도구를](#) 설치합니다. `cargo install cargo-component`

## 권장 도구:

코드 완료 및 문제 해결을 지원하기 위해 IDE에 다음과 같은 선택적 도구를 설치할 수 있습니다.

- rust-analyzer 확장은 [Visual Studio Code의 Rust](#)를 참조하세요.
- Amazon Q Developer에서 [IDE에 Amazon Q Developer 확장 또는 플러그인 설치](#)를 참조하세요.

## 첫 번째 SDK 앱 생성

이 절차에서는 DynamoDB 테이블을 나열하는 첫 번째 SDK for Rust 애플리케이션을 생성합니다.

1. 터미널 또는 콘솔 창에서 앱을 생성하려는 컴퓨터의 위치로 이동합니다.
2. 다음 명령을 실행하여 hello\_world 디렉터리를 생성하고 스켈레톤 Rust 프로젝트로 채웁니다.

```
$ cargo new hello_world --bin
```

3. hello\_world 디렉터리로 이동하여 다음 명령을 사용하여 앱에 필요한 종속성을 추가합니다.

```
$ cargo add aws-config aws-sdk-dynamodb tokio --features tokio/full
```

이러한 종속성에는 비동기 I/O 작업을 구현하는 데 사용되는 크레이트를 포함하여 DynamoDB에 대한 구성 기능과 지원을 제공하는 SDK [tokio 크레이트](#)가 포함됩니다.

### Note

tokio/full Tokio와 같은 기능을 사용하지 않으면 비동기 런타임이 제공되지 않습니다. SDK for Rust에는 비동기 런타임이 필요합니다.

4. 다음 코드를 포함하도록 src 디렉터리main.rs에서를 업데이트합니다.

```
use aws_config::meta::region::RegionProviderChain;
use aws_config::BehaviorVersion;
use aws_sdk_dynamodb::{Client, Error};

/// Lists your DynamoDB tables in the default Region or us-east-1 if a default
/// Region isn't set.
#[tokio::main]
async fn main() -> Result<(), Error> {
```

```

    let region_provider = RegionProviderChain::default_provider().or_else("us-
east-1");
    let config = aws_config::defaults(BehaviorVersion::latest())
        .region(region_provider)
        .load()
        .await;
    let client = Client::new(&config);

    let resp = client.list_tables().send().await?;

    println!("Tables:");

    let names = resp.table_names();

    for name in names {
        println!(" {}", name);
    }

    println!();
    println!("Found {} tables", names.len());

    Ok(())
}

```

### Note

이 예제에서는 결과의 첫 번째 페이지만 표시합니다. 결과의 여러 페이지를 처리하는 방법은 [the section called “페이지 매김”](#) 섹션을 참조하세요.

## 5. 프로그램을 실행합니다:

```
$ cargo run
```

테이블 이름 목록이 표시됩니다.

## 기본 사항

### 사전 조건

를 사용하려면 Rust 및 Cargo가 설치되어 있어야 AWS SDK for Rust합니다.

- Rust 도구 체인 설치: <https://www.rust-lang.org/tools/install>
- 명령을 실행하여 cargo-component [도구를](#) 설치합니다. `cargo install cargo-component`

## 권장 도구:

코드 완료 및 문제 해결을 지원하기 위해 IDE에 다음과 같은 선택적 도구를 설치할 수 있습니다.

- rust-analyzer 확장은 [Visual Studio Code의 Rust](#)를 참조하세요.
- Amazon Q Developer에서 [IDE에 Amazon Q Developer 확장 프로그램 또는 플러그인 설치를 참조하세요.](#)

## Rust 기본 사항

다음은 알아두면 도움이 될 Rust 프로그래밍 언어의 몇 가지 기본 사항입니다. 자세한 내용은 [Rust Programming Language](#)에서 참조합니다.

- Cargo.toml은 표준 Rust 프로젝트 구성 파일이며, 프로젝트에 대한 종속성과 일부 메타데이터를 포함합니다. Rust 소스 파일에는 .rs 파일 확장명이 있습니다. [Hello, Cargo!](#)를 참조하세요.
  - 는 프로파일로 사용자 지정할 Cargo.toml 수 있습니다. [릴리스 프로파일을 사용하여 빌드 사용자 지정을 참조하세요.](#) 이러한 프로파일은 공유 AWS config 파일 내에서 AWS의 프로파일 사용과 완전히 관련이 없으며 독립적입니다.
  - 프로젝트와이 파일에 라이브러리 종속성을 추가하는 일반적인 방법을 사용하는 것입니다 cargo add. [cargo-add](#)을 참조하세요.
- Rust에는 다음과 같은 기본 함수 구조가 있습니다. let 키워드는 변수를 선언하며 할당(=)과 페어링 될 수 있습니다. 이후에 유형을 지정하지 않으면 let 컴파일러가 유형을 유추합니다. [변수 및 변동성을 참조하세요.](#)

```
fn main() {
    let w = "world";
    println!("Hello {}!", w);
}
```

- TRust는 명시적 유형을 x 사용하여 변수를 선언하기 위해 구문을 사용합니다x: T. [데이터 유형을 참조하세요.](#)
- struct X {}는 새 유형을 정의합니다X. 방법은 사용자 지정 구조체 유형에 구현됩니다X. 유형 방법은 키워드 접두사가 붙은 구현 블록으로 선언X됩니다impl. 구현 블록 내에서는 메서드가 호출된 구조체의 인스턴스를 self 나타냅니다. [키워드 impl](#) 및 [메서드 구문](#)을 참조하세요.



- 느낌표("!")가 함수 정의 또는 함수 호출로 보이는 것을 따르는 경우 코드는 매크로를 정의하거나 호출합니다. [매크로를 참조하세요.](#)
- Rust에서는 복구할 수 없는 오류가 panic! 매크로로 표시됩니다. 프로그램이 실행을 중지panic!하고, 실패 메시지를 인쇄하고, 풀고, 스택을 정리하고, 종료합니다. [에서 복구할 수 없는 오류를 참조하세요panic!](#).
- Rust는 다른 프로그래밍 언어와 마찬가지로 기본 클래스의 기능 상속을 지원하지 않습니다. Rusttraits는 메서드의 오버로드 기능을 제공합니다. 특성은 개념적으로 인터페이스와 유사한 것으로 생각할 수 있습니다. 그러나 특성과 실제 인터페이스는 차이가 있으며 설계 프로세스에서 다르게 사용되는 경우가 많습니다. [특성: 공유 동작 정의를 참조하세요.](#)
- 다형성은 각 데이터 유형을 개별적으로 작성할 필요 없이 여러 데이터 유형에 대한 기능을 지원하는 코드를 말합니다. Rust는 열거형, 특성 및 일반을 통해 다형성을 지원합니다. [상속을 유형 시스템으로,를 코드 공유로](#) 참조하세요.
- Rust는 메모리에 대해 매우 명시적입니다. 스마트 포인터 "는 포인터처럼 작동하지만 추가 메타데이터와 기능도 있는 데이터 구조"입니다. [스마트 포인터를 참조하세요.](#)
- 유형은 필요한 경우 호출자에게 메모리 소유권을 이전하는 데 도움이 되는 복제본 clone-on-write 스마트 포인터Cow입니다. [Enum std::borrow::Cow](#)을 참조하세요.
- 유형은 할당된 인스턴스를 계산하는 원자 참조 수 계산 스마트 포인터Arc입니다. [Struct std::sync::Arc](#)을 참조하세요.
- SDK for Rust는 복잡한 유형을 구성하기 위해 빌더 패턴을 자주 사용합니다.

## AWS SDK for Rust crate 기본 사항

- SDK for Rust 라이브러리는 각각 서로 다른 라이브러리 상자로 구분됩니다 AWS 서비스. 이러한 상자는 <https://docs.rs/> 확인할 수 있습니다.
- AWS 서비스 crates는 aws-sdk-s3 및와 aws-sdk-*[servicename]*같은의 명명 규칙을 따릅니다aws-sdk-dynamodb.
- SDK for Rust 기능의 기본 코어 크레이트는 입니다aws-config. 이는 환경에서 구성을 읽을 수 있는 기능을 제공하기 때문에 대부분의 프로젝트에 포함됩니다.
  - 이를 호출 AWS 서비스 되는와 혼동하지 마세요 AWS Config. 이는 서비스이므로 표준 규칙을 따르며 라고 합니다aws-sdk-config.

## 로 작업하기 위한 프로젝트 구성 AWS 서비스

- 애플리케이션에서 사용할 각에 대해 프로젝트에 크레이트를 추가해야 AWS 서비스 합니다.

- 크레이트를 추가하는 권장 방법은 `cargo add [crateName]`등을 실행하여 프로젝트 디렉터리의 명령줄을 사용하는 것입니다 `cargo add aws-sdk-s3`.
- 이렇게 하면에서 프로젝트의 `Cargo.toml`에 줄이 추가됩니다[dependencies].
- 기본적으로이 작업은 최신 버전의 크레이트를 프로젝트에 추가합니다.
- 소스 파일에서 `use` 문을 사용하여 상자의 항목을 범위로 가져옵니다. Rust Programming Language 웹 사이트의 [외부 패키지 사용을 참조하세요](#).
- 크레이트 이름은 하이픈으로 시작하는 경우가 많지만 실제로 크레이트를 사용할 때 하이픈이 밑줄로 변환됩니다. 예를 들어, `aws-config` 크레이트는 코드 `use` 문에 `use aws_config`로 사용됩니다.
- 구성은 복잡한 주제입니다. 구성은 코드에서 직접 수행하거나 환경 변수 또는 구성 파일에서 외부적으로 지정할 수 있습니다. 자세한 내용은 [구성 옵션](#) 단원을 참조하십시오.
- SDK가 구성을 로드하면 대부분의 설정에 합리적인 기본값이 있으므로 실행을 중지하는 대신 잘못된 값이 로깅됩니다. 로깅을 켜는 방법을 알아보려면 섹션을 참조하세요 [AWS SDK for Rust 코드 로깅 활성화](#).
- 대부분의 환경 변수 및 구성 파일 설정은 프로그램이 시작될 때 한 번 로드됩니다. 프로그램을 다시 시작할 때까지 값에 대한 업데이트가 표시되지 않습니다.

## Tokio 런타임

- Tokio는 SDK for Rust 프로그래밍 언어의 비동기 런타임으로 작업을 실행합니다 `async`. [tokio.rs](#) 및 [docs.rs/tokio](#) 참조하십시오.
- SDK for Rust에는 비동기 런타임이 필요합니다. 프로젝트에 다음 크레이트를 추가하는 것이 좋습니다.

```
$ cargo add tokio --features=full
```

- `tokio::main` 속성 매크로는 프로그램에 대한 비동기 기본 진입점을 생성합니다. 이 매크로를 사용하려면 다음과 같이 `main` 메서드 앞의 줄에 매크로를 추가합니다.

```
#[tokio::main]
async fn main() -> Result<(), Error> {
```

# 구성 AWS SDK for Rust

AWS SDK for Rust를 구성하는 방법을 알아봅니다. 를 사용하여 개발할 AWS 때 코드가 인증하는 방법을 설정해야 합니다 AWS 서비스. AWS 리전 사용할 도 설정해야 합니다.

[AWS SDKs 및 도구 참조 안내서](#)에는 많은 AWS SDKs.

## 주제

- [서비스 클라이언트 생성](#)
- [동작 버전](#)
- [자격 증명 공급자](#)
- [AWS 리전 선택](#)
- [구성 옵션](#)
- [클라이언트 엔드포인트](#)
- [HTTP](#)
- [인터셉터](#)
- [관찰 기능 사용](#)
- [클라이언트의 단일 작업 구성 재정의](#)
- [재시도](#)
- [시간 초과](#)

## 서비스 클라이언트 생성

에 요청하려면 먼저 해당 서비스의 클라이언트를 인스턴스화 AWS 서비스합니다. 제한 시간, HTTP 클라이언트 및 재시도 구성과 같은 서비스 클라이언트에 대한 공통 설정을 구성할 수 있습니다.

각 서비스 클라이언트에는 AWS 리전 및 자격 증명 공급자가 필요합니다. SDK는 이러한 값을 사용하여 리소스의 올바른 리전으로 요청을 보내고 올바른 자격 증명으로 요청에 서명합니다. 이러한 값을 프로그래밍 방식으로 코드로 지정하거나 환경에서 자동으로 로드할 수 있습니다.

### Note

서비스 클라이언트는 구성 비용이 많이 들 수 있으며 일반적으로 공유됩니다. 이를 용이하게 Client 하기 위해 모든 구조체는 Clone 구현합니다.

SDK에는 구성 설정 값을 찾기 위해 확인하는 일련의 위치(또는 소스)가 있습니다.

1. 코드나 서비스 클라이언트 자체에 설정된 모든 명시적 설정은 다른 모든 설정보다 우선합니다.
2. 환경 변수
  - 환경 변수 설정에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [환경 변수](#)를 참조하세요. AWS SDKs
3. 공유 config 및 credentials 파일
  - 이러한 파일 설정에 대한 자세한 내용은 SDK [config 및 도구 참조 안내서의 공유 및 credentials 파일을](#) 참조하세요. AWS SDKs
4. SDK 소스 코드 자체에서 제공하는 모든 기본값은 마지막으로 사용됩니다.
  - 리전과 같은 일부 속성에는 기본값이 없습니다. 코드, 환경 설정 또는 공유 config 파일에 명시적으로 지정해야 합니다. SDK가 필요한 구성을 해결할 수 없는 경우 런타임에 API 요청이 실패할 수 있습니다.

대부분의 환경 변수 설정과 config 및 credentials 파일 설정은 일관된 동작을 위해 다중 AWS SDKs 및 도구에서 공유됩니다. 환경 변수 또는 구성 파일에서 SDK가 확인할 수 있는 모든 설정을 보려면 SDK 및 도구 참조 안내서의 [설정](#) 참조를 참조하세요. AWS SDKs

## 환경에서 클라이언트 구성

환경 기반 구성으로 클라이언트를 생성하려면 aws-config 크레딧의 정적 메서드를 사용합니다.

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);
```

이러한 방식으로 클라이언트를 생성하면 Amazon Elastic Compute Cloud AWS Lambda 또는 서비스 클라이언트의 구성을 환경에서 직접 사용할 수 있는 기타 컨텍스트에서 실행할 때 유용합니다. 이렇게 하면 실행 중인 환경에서 코드를 분리하여 코드를 변경 AWS 리전 하지 않고도 애플리케이션을 여러에 더 쉽게 배포할 수 있습니다.

특정 속성을 명시적으로 재정의할 수 있습니다. 명시적 구성은 실행 환경에서 확인된 구성보다 우선합니다. 다음 예제에서는 환경에서 구성을 로드하지만 명시적으로 재정의합니다. AWS 리전

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .region("us-east-1")
```

```

    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);

```

### Note

생성 시 클라이언트가 모든 구성 값을 소싱하지는 않습니다. 임시 액세스 키 및 IAM Identity Center 구성과 같은 자격 증명 관련 설정은 클라이언트를 사용하여 요청할 때 자격 증명 공급자 계층에서 액세스합니다.

이전 예제에 `BehaviorVersion::latest()` 표시된 코드는 기본값에 사용할 SDK 버전을 나타냅니다. `BehaviorVersion::latest()`는 대부분의 경우에 적합합니다. 세부 정보는 [동작 버전](#)을 참조하세요.

## 서비스별 설정에 빌더 패턴 사용

특정 서비스 클라이언트 유형에만 구성할 수 있는 몇 가지 옵션이 있습니다. 그러나 대부분의 경우 환경에서 대부분의 구성을 로드한 다음 특히 추가 옵션을 추가해야 합니다. 빌더 패턴은 AWS SDK for Rust 상자 내의 일반적인 패턴입니다. 먼저를 사용하여 일반 구성을 로드한 `aws_config::defaults` 다음 `from` 메서드를 사용하여 작업 중인 서비스의 빌더에 해당 구성을 로드합니다. 그런 다음 해당 서비스에 대한 고유한 구성 값을 설정하고를 호출할 수 있습니다 `build`. 마지막으로 클라이언트는 이 수정된 구성에서 생성됩니다.

```

// Call a static method on aws-config that sources default config values.
let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

// Use the Builder for S3 to create service-specific config from the default config.
let s3_config = aws_sdk_s3::config::Builder::from(&config)
    .accelerate(true) // Set an S3-only configuration option
    .build();

// Create the client.
let s3 = aws_sdk_s3::Client::from_conf(s3_config);

```

특정 유형의 서비스 클라이언트에 사용할 수 있는 추가 메서드를 검색하는 한 가지 방법은 옹와 같은 API 설명서를 사용하는 것입니다 [aws\\_sdk\\_s3::config::Builder](#).

## 고급 명시적 클라이언트 구성

환경에서 구성을 로드하는 대신 특정 값으로 서비스 클라이언트를 구성하려면 다음과 같이 클라이언트 Config 빌더에서 해당 값을 지정할 수 있습니다.

```
let conf = aws_sdk_s3::Config::builder()
    .region("us-east-1")
    .endpoint_resolver(my_endpoint_resolver)
    .build();

let s3 = aws_sdk_s3::Client::from_conf(conf);
```

를 사용하여 서비스 구성을 생성하면 기본 구성이 로드 `aws_sdk_s3::Config::builder()` 되지 않습니다. 기본값은 `aws_config::defaults`를 기반으로 구성을 생성할 때만 로드됩니다.

특정 서비스 클라이언트 유형에만 구성할 수 있는 몇 가지 옵션이 있습니다. 이전 예제에서는 Amazon S3 클라이언트에서 `endpoint_resolver` 함수를 사용하여 이 예제를 보여줍니다.

## 동작 버전

AWS SDK for Rust 개발자는 언어와 주요 라이브러리가 제공하는 강력하고 예측 가능한 동작을 기대하고 신뢰합니다. SDK for Rust를 사용하는 개발자가 예상 동작을 얻을 수 있도록 하려면 클라이언트 구성에 `BehaviorVersion`가 포함되어야 합니다. `BehaviorVersion`는 기본값이 예상되는 SDK의 버전을 지정합니다. 이렇게 하면 시간이 지남에 따라 SDK가 발전하여 애플리케이션 동작에 예기치 않은 악영향 없이 새로운 표준에 맞게 모범 사례를 변경하고 새로운 기능을 지원할 수 있습니다.

### Warning

를 명시적으로 지정하지 않고 SDK를 구성하거나 클라이언트를 생성하려고 하면 `BehaviorVersion` 생성자가 `panic`를 수행합니다.

예를 들어 SDK의 새 버전이 새 기본 재시도 정책과 함께 릴리스되었다고 가정해 보겠습니다. 애플리케이션이 SDK의 이전 버전과 `BehaviorVersion` 일치하는 것을 사용하는 경우 새 기본 구성 대신 이전 구성이 사용됩니다.

SDK for Rust의 새 동작 버전이 릴리스될 때마다 이전 버전 `BehaviorVersion`에는 SDK for Rust `deprecated` 속성이 표시되고 새 버전이 추가됩니다. 이렇게 하면 컴파일 시 경고가 발생하지만, 그렇

지 않으면 평소처럼 빌드를 계속할 수 있습니다. `BehaviorVersion::latest()`도 새 버전의 기본 동작을 나타내도록 업데이트됩니다.

### Note

대부분의 경우 코드 `BehaviorVersion::latest()` 또는 `Cargo.toml` 파일의 기능 플래그 `behavior-version-latest`에를 사용해야 합니다. 가 필요한 경우에만 특정 버전에 고정할 것을 권장합니다.

## 에서 동작 버전 설정 `Cargo.toml`

`Cargo.toml` 파일에 적절한 기능 플래그를 `aws-sdk-iam` 포함하여 SDK 및 `aws-sdk-s3` 또는 와 같은 개별 모듈의 동작 버전을 지정할 수 있습니다. 현재에서는 SDK latest 버전만 지원됩니다 `Cargo.toml`.

```
[dependencies]
aws-config = { version = "1", features = ["behavior-version-latest"] }
aws-sdk-s3 = { version = "1", features = ["behavior-version-latest"] }
```

## 코드에서 동작 버전 설정

SDK 또는 클라이언트를 구성할 때 코드를 지정하여 필요에 따라 동작 버전을 변경할 수 있습니다.

```
let config = aws_config::load_defaults(BehaviorVersion::v2023_11_09()).await;
```

이 예제에서는 환경을 사용하여 SDK를 구성하지만 `BehaviorVersion`로 설정하는 구성을 생성합니다 `v2023_11_09()`.

## 자격 증명 공급자

를 AWS 사용하여 요청을 하기 위해 SDK는 AWS SDK for Rust에서 발급한 암호화 서명 보안 인증을 사용합니다 AWS. 런타임 시 SDK는 여러 위치를 확인하여 자격 증명의 구성 값을 검색합니다.

검색된 구성에 [AWS IAM Identity Center Single Sign-On 액세스 설정](#)이 포함된 경우 SDK는 IAM Identity Center와 협력하여 요청에 사용하는 임시 자격 증명을 검색합니다 AWS 서비스.

검색된 구성에 [임시 자격 증명](#)이 포함된 경우 SDK는 이를 사용하여 AWS 서비스 호출합니다. 임시 자격 증명은 액세스 키와 세션 토큰으로 구성됩니다.

를 사용한 인증은 코드베이스 외부에서 처리할 AWS 수 있습니다. SDK는 자격 증명 공급자 체인을 사용하여 많은 인증 방법을 자동으로 감지, 사용 및 새로 고칠 수 있습니다.

프로젝트의 AWS 인증을 시작하기 위한 안내 옵션은 AWS SDKs 및 도구 참조 안내서의 [인증 및 액세스](#)를 참조하세요.

## 자격 증명 공급자 체인

클라이언트를 구성할 때 자격 증명 공급자를 명시적으로 지정하지 않으면 SDK for Rust는 자격 증명을 제공할 수 있는 일련의 위치를 확인하는 자격 증명 공급자 체인을 사용합니다. SDK가 이러한 위치 중 하나에서 자격 증명을 찾으면 검색이 중지됩니다. 클라이언트 구성에 대한 자세한 내용은 섹션을 참조하세요 [서비스 클라이언트 생성](#).

다음 예제에서는 코드에 자격 증명 공급자를 지정하지 않습니다. SDK는 자격 증명 공급자 체인을 사용하여 호스팅 환경에 설정된 인증을 감지하고 호출에 해당 인증을 사용합니다 AWS 서비스.

```
let config = aws_config::defaults(BehaviorVersion::latest()).load().await;
let s3 = aws_sdk_s3::Client::new(&config);
```

## 자격 증명 검색 순서

자격 증명 공급자 체인은 다음과 같은 사전 정의된 시퀀스를 사용하여 자격 증명을 검색합니다.

### 1. 액세스 키 환경 변수

SDK는 AWS\_ACCESS\_KEY\_ID 및 환경 AWS\_SECRET\_ACCESS\_KEY AWS\_SESSION\_TOKEN 변수에서 자격 증명을 로드하려고 시도합니다.

### 2. 공유 AWS **config** 및 **credentials** 파일

SDK는 공유 AWS config 및 credentials 파일의 [default] 프로필에서 자격 증명을 로드하려고 시도합니다. AWS\_PROFILE 환경 변수를 사용하여 사용하는 대신 SDK가 로드할 명명된 프로파일을 선택할 수 있습니다 [default]. config 및 credentials 파일은 다양한 AWS SDKs 및 도구에서 공유됩니다. 이러한 파일에 대한 자세한 내용은 SDK [config 및 도구 참조 안내서의 공유 및 credentials 파일을](#) 참조하세요. AWS SDKs

IAM Identity Center를 사용하여 인증하는 경우 SDK for Rust가 AWS CLI 명령을 실행하여 설정한 Single Sign-On 토큰을 사용하는 경우입니다 `aws sso login`. SDK는 IAM Identity Center가 유효한 토큰으로 교환한 임시 자격 증명을 사용합니다. 그러면 SDK는 호출 시 임시 자격 증명을 사용합니다 AWS 서비스. 이 프로세스에 대한 자세한 내용은 [SDK 및 도구 참조 안내서의에 대한 SDK 자격 증명 확인 이해 AWS 서비스](#) AWS SDKs.



- 이 공급자 구성에 대한 지침은 SDK 및 도구 참조 안내서의 [IAM Identity Center 인증](#)을 참조하세요. AWS SDKs
- 이 공급자의 SDK 구성 속성에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [IAM Identity Center 자격 증명 공급자](#)를 참조하세요. AWS SDKs

### 3. AWS STS 웹 자격 증명

액세스가 필요한 모바일 애플리케이션 또는 클라이언트 기반 웹 애플리케이션을 생성할 때 AWS AWS Security Token Service (AWS STS)는 퍼블릭 자격 증명 공급자(IdP)를 통해 인증된 페더레이션 사용자를 위한 임시 보안 자격 증명 세트를 반환합니다.

- 프로필에서 이를 지정하면 SDK 또는 도구가 AssumeRoleWithWebIdentity API 메서드를 사용하여 AWS STS 임시 자격 증명을 검색하려고 시도합니다. 이 방법에 대한 자세한 내용은 AWS Security Token Service API 참조의 [AssumeRoleWithWebIdentity](#)를 참조하세요.
- 이 공급자 구성에 대한 지침은 SDK 및 도구 참조 안내서의 [웹 자격 증명 또는 OpenID Connect를 사용한 페더레이션](#)을 참조하세요. AWS SDKs
- 이 공급자의 SDK 구성 속성에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [역할 자격 증명 공급자 수입](#)을 참조하세요. AWS SDKs

### 4. Amazon ECS 및 Amazon EKS 컨테이너 자격 증명

Amazon Elastic Container Service 태스크와 Kubernetes 서비스 계정에는 연결된 IAM 역할이 있을 수 있습니다. IAM 역할에서 부여된 권한은 포드의 작업 또는 컨테이너에서 실행되는 컨테이너에 의해 수입됩니다. 이 역할을 사용하면 SDK for Rust 애플리케이션 코드(컨테이너에 있음)가 다른를 사용할 수 있습니다 AWS 서비스.

SDK는 Amazon ECS 및 Amazon EKS에서 자동으로 설정할 수 있는 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` 또는 `AWS_CONTAINER_CREDENTIALS_FULL_URI` 환경 변수에서 자격 증명을 검색하려고 시도합니다.

- Amazon ECS에이 역할을 설정하는 방법에 대한 자세한 내용은 [Amazon Elastic Container Service 개발자 안내서의 Amazon ECS 태스크 IAM 역할을](#) 참조하세요.
- Amazon EKS 설정 정보는 [Amazon EKS 사용 설명서의 Amazon EKS Pod Identity Agent 설정](#)을 참조하세요.
- 이 공급자의 SDK 구성 속성에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [컨테이너 자격 증명 공급자](#)를 참조하세요. AWS SDKs

### 5. Amazon EC2 인스턴스 메타데이터 서비스

IAM 역할을 생성하고 인스턴스에 연결합니다. 인스턴스의 SDK for Rust 애플리케이션은 인스턴스 메타데이터에서 역할이 제공하는 자격 증명을 검색하려고 시도합니다.

- SDK for Rust는 [IMDSv2](#)만 지원합니다.
  - 이 역할을 설정하고 메타데이터, [Amazon EC2에 대한 IAM 역할](#) 및 [인스턴스 메타데이터 작업](#)에 대한 자세한 내용은 Amazon EC2 사용 설명서에서 확인하세요.
  - 이 공급자의 SDK 구성 속성에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [IMDS 자격 증명 공급자](#)를 참조하세요. AWS SDKs
6. 이 시점에서 자격 증명이 여전히 해결되지 않으면 오류가 panics 있는 작업입니다.

AWS 자격 증명 공급자 구성 설정에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 설정 참조에서 [표준화된 자격 증명 공급자](#)를 참조하세요. AWS SDKs

## 명시적 자격 증명 공급자

자격 증명 공급자 체인에 의존하여 인증 방법을 감지하는 대신 SDK에서 사용해야 하는 특정 자격 증명 공급자를 지정할 수 있습니다. 를 사용하여 일반 구성을 로드aws\_config::defaults할 때 다음과 같이 사용자 지정 자격 증명 공급자를 지정할 수 있습니다.

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .credentials_provider(MyCredentialsProvider::new())
    .load()
    .await;
```

[ProvideCredentials](#) 특성을 구현하여 자체 자격 증명 공급자를 구현할 수 있습니다.

## 자격 증명 캐싱

SDK는 자격 증명 및 SSO 토큰과 같은 기타 자격 증명 유형을 캐싱합니다. 기본적으로 SDK는 첫 번째 요청 시 자격 증명을 로드하고 캐싱한 다음 만료에 가까워지면 다른 요청 중에 자격 증명을 새로 고치려고 시도하는 지연 캐시 구현을 사용합니다. 동일한에서 생성된 클라이언트SdkConfig를 공유합니다[IdentityCache](#).

## AWS 리전 선택

를 사용하여 특정 지리적 영역에서 AWS 서비스 작동하는에 액세스할 수 있습니다 AWS 리전. 이는 중복성과 데이터 및 애플리케이션을 사용자와 사용자가 액세스하는 위치와 가깝게 실행하는 데 모두 유용할 수 있습니다. 리전 사용 방법에 대한 자세한 내용은 SDK 및 도구 참조 안내서[AWS 리전](#)의 섹션을 참조하세요. AWS SDKs

**⚠ Important**

대부분의 리소스는 특정에 상주 AWS 리전 하며 SDK를 사용할 때 리소스에 대한 올바른 리전을 제공해야 합니다.

## AWS 리전 공급자 체인

다음 조회 프로세스는 실행 환경에서 서비스 클라이언트의 구성을 로드할 때 사용됩니다. SDK에서 발견한 첫 번째 값은 클라이언트 구성에 사용됩니다. 서비스 클라이언트 생성에 대한 자세한 내용은 섹션을 참조하세요 [환경에서 클라이언트 구성](#).

1. 프로그래밍 방식으로 설정된 명시적 리전.
2. `AWS_REGION` 환경 변수를 확인합니다.
  - AWS Lambda 서비스를 사용하는 경우 이 환경 변수는 AWS Lambda 컨테이너에 의해 자동으로 설정됩니다.
3. 공유 AWS config 파일의 region 속성이 확인됩니다.
  - `AWS_CONFIG_FILE` 환경 변수를 사용하여 공유 config 파일의 위치를 변경할 수 있습니다. 이 파일이 보관되는 위치에 대한 자세한 내용은 SDK [config 및 도구 참조 안내서의 공유 및 credentials 파일의 위치를](#) 참조하세요. AWS SDKs
  - `AWS_PROFILE` 환경 변수를 사용하여 기본값 대신 명명된 프로파일을 선택할 수 있습니다. 다양한 프로필 구성에 대한 자세한 내용은 SDK [config 및 도구 참조 안내서의 공유 및 credentials 파일을](#) 참조하세요. AWS SDKs
4. SDK는 Amazon EC2 인스턴스 메타데이터 서비스를 사용하여 현재 실행 중인 Amazon EC2 인스턴스의 리전을 확인하려고 시도합니다.
  - 는 IMDSv2 AWS SDK for Rust 만 지원합니다.

## 구성 옵션

코드 외부에서 많은 구성 설정을 처리할 수 있습니다. 대부분의 구성 설정은 환경 변수 또는 별도의 공유 AWS config 파일로 설정할 수 있습니다. 공유 config 파일은 프로필이라는 별도의 설정 세트를 유지하여 다양한 환경 또는 테스트에 대해 다양한 구성을 제공할 수 있습니다.

환경 변수 및 공유 config 파일 설정은 표준화되고 AWS SDKs 및 도구 간에 공유되어 다양한 언어에서 일관된 기능을 지원합니다.

AWS SDKs 및 도구 참조 안내서를 참조하여 이러한 방법을 통해 애플리케이션을 구성하는 방법과 각 크로스 SDK 설정에 대한 세부 정보를 알아봅니다. 일부 관심 페이지는 다음과 같습니다.

- [공유 config 및 credentials 파일](#) - 이러한 구성 파일의 프로필과 형식을 설명합니다.
- [공유 config 및 credentials 파일의 위치](#) - 이러한 파일의 기본 위치와 변경 방법을 설명합니다.
- [환경 변수 지원](#) - 환경 변수 설정을 설명합니다.
- [참조 설정](#) - 모든 교차 SDK 설정에 대한 참조 정보입니다.

## 클라이언트 엔드포인트

가을 AWS SDK for Rust 호출할 때 첫 번째 단계 AWS 서비스중 하나는 요청을 라우팅할 위치를 결정하는 것입니다. 이 프로세스를 엔드포인트 해상도라고 합니다.

서비스 클라이언트를 생성할 때 SDK에 대한 엔드포인트 확인을 구성할 수 있습니다. 엔드포인트 확인의 기본 구성은 일반적으로 좋지만 기본 구성을 수정하려는 몇 가지 이유가 있습니다. 두 가지 이유는 다음과 같습니다.

- 서비스의 사전 릴리스 버전 또는 서비스의 로컬 배포를 요청하려면
- SDK에서 아직 모델링되지 않은 특정 서비스 기능에 액세스하려면

### Warning

엔드포인트 확인은 고급 SDK 주제입니다. 기본 설정을 변경하면 코드가 해제될 위험이 있습니다. 기본 설정은 프로덕션 환경의 대부분의 사용자에게 적용됩니다.

사용자 지정 엔드포인트는 모든 서비스 요청에 사용되도록 전역적으로 설정하거나 특성에 대해 사용자 지정 엔드포인트를 설정할 수 있습니다 AWS 서비스.

사용자 지정 엔드포인트는 공유 AWS config 파일의 환경 변수 또는 설정을 사용하여 구성할 수 있습니다. 이 접근 방식에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [서비스별 엔드포인트](#)를 참조하세요. AWS SDKs 모든에 대한 공유 config 파일 설정 및 환경 변수의 전체 목록은 서비스별 엔드포인트 식별자를 AWS 서비스참조하세요. <https://docs.aws.amazon.com/sdkref/latest/guide/ss-endpoints-table.html>

또는 다음 섹션과 같이 코드에서 사용자 지정을 구성할 수도 있습니다.

## 사용자 지정 구성

클라이언트를 빌드할 때 사용할 수 있는 두 가지 방법으로 서비스 클라이언트의 엔드포인트 확인을 사용자 지정할 수 있습니다.

1. `endpoint_url(url: Into<String>)`
2. `endpoint_resolver(resolver: impl crate::config::endpoint::ResolveEndpoint + `static)`

두 속성을 모두 설정할 수 있습니다. 그러나 대부분 하나만 제공합니다. 일반적인 사용의 경우 `endpoint_url`는 가장 자주 사용자 지정됩니다.

### 엔드포인트 URL 설정

에 대한 값을 설정 `endpoint_url`하여 서비스의 "기본" 호스트 이름을 나타낼 수 있습니다. 그러나 이 값은 클라이언트 `ResolveEndpoint` 인스턴스에 파라미터로 전달되므로 최종적인 것은 아닙니다. 그런 다음 `ResolveEndpoint` 구현은 해당 값을 검사하고 잠재적으로 수정하여 최종 엔드포인트를 결정할 수 있습니다.

### 엔드포인트 해석기 설정

서비스 클라이언트의 `ResolveEndpoint` 구현에 따라 SDK가 지정된 요청에 사용하는 최종 해결 엔드포인트가 결정됩니다. 서비스 클라이언트는 모든 요청에 대해 `resolve_endpoint` 메서드를 호출하고 더 이상 변경하지 않고 해석기가 반환하는 [EndpointFuture](#) 값을 사용합니다.

다음 예제에서는 스테이징 및 프로덕션과 같은 단계당 다른 엔드포인트를 해결하는 Amazon S3 클라이언트에 대한 사용자 지정 엔드포인트 해석기 구현을 제공하는 방법을 보여줍니다.

```
use aws_sdk_s3::config::endpoint::{ResolveEndpoint, EndpointFuture, Params, Endpoint};

#[derive(Debug)]
struct StageResolver { stage: String }
impl ResolveEndpoint for StageResolver {
    fn resolve_endpoint(&self, params: &Params) -> EndpointFuture<'_> {
        let stage = &self.stage;
        EndpointFuture::ready(Ok(Endpoint::builder().url(format!(
            "{stage}.myservice.com")).build()))
    }
}
```

```

let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

let resolver = StageResolver { stage: std::env::var("STAGE").unwrap() };

let s3_config = aws_sdk_s3::config::Builder::from(&config)
    .endpoint_resolver(resolver)
    .build();

let s3 = aws_sdk_s3::Client::from_conf(s3_config);

```

### Note

엔드포인트 해석기 및 확장을 통해 ResolveEndpoint 특성은 각 서비스에 고유하므로 서비스 클라이언트 구성에서만 구성할 수 있습니다. 반면 엔드포인트 URL은 공유 구성을 사용하거나(동에서 파생된 모든 서비스에 적용) 특정 서비스에 대해 구성할 수 있습니다.

## ResolveEndpoint 파라미터

resolve\_endpoint 메서드는 엔드포인트 확인에 사용되는 속성을 포함하는 서비스별 파라미터를 허용합니다.

모든 서비스에는 다음과 같은 기본 속성이 포함됩니다.

명칭	유형	설명
region	String	클라이언트의 AWS 리전
endpoint	String	값 집합의 문자열 표현 endpointUrl
use_fips	불	클라이언트 구성에서 FIPS 엔드포인트가 활성화되어 있는지 여부
use_dual_stack	불	클라이언트 구성에서 듀얼 스택 엔드포인트가 활성화되었는지 여부

AWS 서비스는 해상도에 필요한 추가 속성을 지정할 수 있습니다. 예를 들어 Amazon S3 [엔드포인트 파라미터](#)에는 버킷 이름과 여러 Amazon S3-specific 기능 설정이 포함됩니다. 예를 들어 `force_path_style` 속성은 가상 호스트 주소 지정을 사용할 수 있는지 여부를 결정합니다.

자체 공급자를 구현하는 경우 엔드포인트 파라미터의 자체 인스턴스를 구성할 필요가 없습니다. SDK는 각 요청에 대한 속성을 제공하고 이를 구현에 전달합니다 `resolve_endpoint`.

## 를 사용한 `endpoint_url`와를 사용한 비교 `endpoint_resolver`

다음 두 가지 구성, 즉를 사용하는 구성 `endpoint_url`과를 사용하는 구성은 동등한 엔드포인트 확인 동작을 가진 클라이언트를 생성하지 `endpoint_resolver`않는다는 점을 이해하는 것이 중요합니다.

```
use aws_sdk_s3::config::endpoint::{ResolveEndpoint, EndpointFuture, Params, Endpoint};

#[derive(Debug, Default)]
struct CustomResolver;
impl ResolveEndpoint for CustomResolver {
    fn resolve_endpoint(&self, _params: &Params) -> EndpointFuture<'_> {
        EndpointFuture::ready(Ok(Endpoint::builder().url("https://
endpoint.example").build()))
    }
}

let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

// use endpoint url
aws_sdk_s3::config::Builder::from(&config)
    .endpoint_url("https://endpoint.example")
    .build();

// Use endpoint resolver
aws_sdk_s3::config::Builder::from(&config)
    .endpoint_resolver(CustomResolver::default())
    .build();
```

를 설정하는 클라이언트는 엔드포인트 확인의 일부로 수정할 수 있는 (기본) 공급자에 전달되는 기본 URL을 `endpoint_url` 지정합니다.

를 설정하는 클라이언트는 Amazon S3 클라이언트가 사용하는 최종 URL을 `endpoint_resolver` 지정합니다.

## 예시

사용자 지정 엔드포인트는 종종 테스트에 사용됩니다. 클라우드 기반 서비스에 직접 호출하는 대신 로컬에서 호스팅되고 시뮬레이션된 서비스로 호출이 라우팅됩니다. 이러한 두 가지 옵션은 다음과 같습니다.

- [DynamoDB 로컬](#) - Amazon DynamoDB 서비스의 로컬 버전입니다.
- [LocalStack](#) - 로컬 시스템의 컨테이너에서 실행되는 클라우드 서비스 에뮬레이터입니다.

다음 예제에서는 이러한 두 테스트 옵션을 사용할 사용자 지정 엔드포인트를 지정하는 두 가지 방법을 보여줍니다.

### 코드에서 DynamoDB 로컬 직접 사용

이전 단원에서 설명한 대로 코드에서 `endpoint_url` 직접을 설정하여 로컬 DynamoDB 서버를 가리키도록 기본 엔드포인트를 재정의할 수 있습니다. 코드에서:

```
let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
    .test_credentials()
    // DynamoDB run locally uses port 8000 by default.
    .endpoint_url("http://localhost:8000")
    .load()
    .await;
let dynamodb_local_config =
aws_sdk_dynamodb::config::Builder::from(&config).build();

let client = aws_sdk_dynamodb::Client::from_conf(dynamodb_local_config);
```

[전체 예제](#)는 GitHub에서 확인할 수 있습니다.

### config 파일을 사용하여 LocalStack 사용

공유 AWS config 파일에서 [서비스별 엔드포인트](#)를 설정할 수 있습니다. 다음 구성 프로파일은 포트 localhost에서 `endpoint_url`하도록 설정됩니다. LocalStack 구성에 대한 자세한 내용은 [LocalStack 문서 웹 사이트의 엔드포인트 URL을 통한 LocalStack 액세스](#)를 참조하세요.

```
[profile localstack]
region=us-east-1
endpoint_url = http://localhost:4566
```



SDK는 공유 config 파일의 변경 사항을 선택하여 localstack 프로필을 사용할 때 SDK 클라이언트에 적용합니다. 이 접근 방식을 사용하면 코드가 엔드포인트에 대한 참조를 포함할 필요가 없으며 다음과 같습니다.

```
// set the environment variable `AWS_PROFILE=localstack` when running
// the application to source `endpoint_url` and point the SDK at the
// localstack instance
let config = aws_config::defaults(BehaviorVersion::latest()).load().await;

let s3_config = aws_sdk_s3::config::Builder::from(&config)
    .force_path_style(true)
    .build();

let s3 = aws_sdk_s3::Client::from_conf(s3_config);
```

[전체 예제](#)는 GitHub에서 확인할 수 있습니다.

## HTTP

AWS SDK for Rust는 다음과 같은 HTTP 관련 구성을 제공합니다.

주제

- [하이퍼 1.x](#)

### 하이퍼 1.x

#### Note

는 안정적인 1.0 버전을 hyper 릴리스하기 전에 개발 AWS SDK for Rust 되었습니다. SDK는 현재의 v0.14 버전 사용으로 기본 설정되어 있습니다hyper.

hyper는와 함께 API 서비스 호출 AWS SDK for Rust 에 사용할 수 있는 Rust용 하위 수준 HTTP 라이브러리입니다. 이 크레딧에 대한 참조 설명서는 [하이퍼 - Rust](#)를 참조하세요. 기본적으로 SDK for Rust는 hyper 및 CryptoMode:Ring 공급자의 v0.14 버전을 사용합니다.

hyper 1.0을 지원하는 클라이언트는 [aws-smithy-experimental](#) 크레딧에 있습니다.

## 암호화 공급자 선택

를 선택하고 `aws-smithy-experimental` 크레이트에서 필요한 기능을 `CryptoProvider` 활성화해야 합니다.

다음 `CryptoProvider` 모드를 사용할 수 있습니다.

- `CryptoMode:AwsLc` - 기반 암호화 공급자입니다 [aws-lc-rs](#). 이 라이브러리는 Google BoringSSL 프로젝트 및 OpenSSL 프로젝트의 코드를 기반으로 AWS 및 해당 고객을 위해 AWS 암호화 팀이 관리하는 범용 암호화 라이브러리입니다.
  - 기능이 필요합니다 `crypto-aws-lc`.
- `CryptoMode:AwsLcFips` -를 기반으로 하는 FIPS 준수 암호화 공급자입니다 [aws-lc-rs](#).
  - 기능이 필요합니다 `crypto-aws-lc-fips`.

### Note

이렇게 하려면 추가 빌드 도구가 필요할 수 있습니다. 자세한 내용은 리 [aws-lc-rs](#) 포지토리 및 빌드 지침을 참조하세요.

- `CryptoMode:Ring` - 기반 암호화 공급자입니다 [ring](#). 이 암호는 "Rust를 사용하는 안전하고 빠른 소형 암호"입니다.
  - 기능이 필요합니다 `crypto-ring`.

### Note

SDK for Rust가 기본 `hyper v0.14` 클라이언트에서 사용하는 암호화 공급자입니다.

크레이트에 대한 기능을 활성화하는 방법

다음 예제에서는 프로젝트의 `aws-smithy-experimental` 크레이트에 `crypto-aws-lc` 기능을 추가하는 방법을 보여줍니다. `Cargo.toml`

```
[dependencies]
aws-smithy-experimental = { version = "0.1.3", features = ["crypto-aws-lc"] }
```

또는 프로젝트 폴더의 명령 프롬프트에서 다음 명령을 실행할 수 있습니다.

```
$ cargo add aws-smithy-experimental -F crypto-aws-lc
```

자세한 내용은 카고 북의 [기능을](#) 참조하세요.

## hyper 1.0 클라이언트를 생성하고 SDK와 함께 사용

hyper 1.0 클라이언트를 생성하려면 프로젝트의 Cargo.toml에 aws-smithy-experimental 크레이트를 추가합니다. HyperClientBuilder 사용하려면 선택해야 합니다 CryptoProvider. 이전 섹션에서는 공급자 선택 사항과 aws-smithy-experimental 크레이트의 암호화 관련 기능 중 하나를 활성화하는 방법을 다루었습니다.

다음 코드는 hyper를 Amazon Simple Storage Service(Amazon S3) 서비스 클라이언트와 함께 사용할 1.0 HTTP 클라이언트의 암호화 공급자CryptoMode::AwsLc로 사용합니다. 이 예제에서는 프로젝트의 에도 AWS 서비스의 상자가 포함되어야 Cargo.toml 합니다aws-sdk-s3.

```
use aws-smithy-experimental::hyper_1_0::{ CryptoMode, HyperClientBuilder };
use aws_smithy_runtime_api::client::behavior_version::BehaviorVersion;

let http_client = HyperClientBuilder::new()
    .crypto_mode(CryptoMode::AwsLc) // Choose a crypto provider.
    .build_https();

let config = aws_config::defaults(BehaviorVersion::latest())
    .http_client(http_client) // Set the http_client on the shared config struct.
    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);
```

HTTP 클라이언트는 공유 구성 구조 또는 특정 서비스의 구성 구조에서 설정할 수 있습니다.

## 인터셉터

인터셉터를 사용하여 API 요청 및 응답 실행에 연결할 수 있습니다. 인터셉터는 SDK가 요청/응답 수명 주기에 동작을 주입하기 위해 작성하는 코드를 호출하는 개방형 메커니즘입니다. 이렇게 하면 진행 중인 요청을 수정하고, 요청 처리를 디버그하고, 오류를 보는 등의 작업을 수행할 수 있습니다.

다음 예제는 재시도 루프가 입력되기 전에 모든 발신 요청에 헤더를 추가하는 간단한 인터셉터를 보여줍니다.

```

use std::borrow::Cow;
use aws_smithy_runtime_api::client::interceptors::{
    Intercept,
    context::BeforeTransmitInterceptorContextMut,
};
use aws_smithy_runtime_api::client::runtime_components::RuntimeComponents;
use aws_smithy_types::config_bag::ConfigBag;
use aws_smithy_runtime_api::box_error::BoxError;

#[derive(Debug)]
struct AddHeaderInterceptor {
    key: Cow<'static, str>,
    value: Cow<'static, str>,
}

impl AddHeaderInterceptor {
    fn new(key: &'static str, value: &'static str) -> Self {
        Self {
            key: Cow::Borrowed(key),
            value: Cow::Borrowed(value),
        }
    }
}

impl Intercept for AddHeaderInterceptor {
    fn name(&self) -> &'static str {
        "AddHeader"
    }

    fn modify_before_retry_loop(
        &self,
        context: &mut BeforeTransmitInterceptorContextMut<'_,>,
        _runtime_components: &RuntimeComponents,
        _cfg: &mut ConfigBag,
    ) -> Result<(), BoxError> {
        let headers = context.request_mut().headers_mut();
        headers.insert(self.key.clone(), self.value.clone());

        Ok(())
    }
}

```

자세한 내용과 사용 가능한 인터셉터 후크는 [인터셉트](#) 특성을 참조하세요.

## 인터셉터 등록

서비스 클라이언트를 구성하거나 특정 작업에 대한 구성을 재정의할 때 인터셉터를 등록합니다. 이 등록은 인터셉터가 클라이언트의 모든 작업에 적용할지 아니면 특정 작업에만 적용할지에 따라 다릅니다.

### 서비스 클라이언트의 모든 작업에 대한 인터셉터

전체 클라이언트에 대한 인터셉터를 등록하려면 Builder 패턴을 사용하여 인터셉터를 추가합니다.

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

// All service operations invoked using 's3' will have the header added.
let s3_conf = aws_sdk_s3::config::Builder::from(&config)
    .interceptor(AddHeaderInterceptor::new("x-foo-version", "2.7"))
    .build();

let s3 = aws_sdk_s3::Client::from_conf(s3_conf);
```

### 특정 작업 전용 인터셉터

단일 작업에 대해서만 인터셉터를 등록하려면 `customize` 확장을 사용합니다. 이 방법을 사용하여 작업별 수준에서 서비스 클라이언트 구성을 재정의할 수 있습니다. 사용자 지정 가능한 작업에 대한 자세한 내용은 [섹션을 참조하세요](#) 클라이언트의 단일 작업 구성 재정의.

```
// Only the list_buckets operation will have the header added.
s3.list_buckets()
    .customize()
    .interceptor(AddHeaderInterceptor::new("x-bar-version", "3.7"))
    .send()
    .await?;
```

## 관찰 기능 사용

관찰성은 시스템의 현재 상태를 내보내는 데이터에서 추론할 수 있는 정도입니다. 방출되는 데이터를 일반적으로 원격 측정이라고 합니다.

주제

- [AWS SDK for Rust 코드 로깅 활성화](#)

## AWS SDK for Rust 코드 로깅 활성화

는 로깅에 [추적](#) 프레임워크를 AWS SDK for Rust 사용합니다.

1. 프로젝트 디렉터리에 대한 명령 프롬프트에서 [추적 구독자](#) 크레인을 종속 항목으로 추가합니다.

```
$ cargo add tracing-subscriber --features tracing-subscriber/env-filter
```

그러면 Cargo.toml 파일의 [dependencies] 섹션에 크레인이 추가됩니다.

2. 구독자를 초기화합니다. 일반적으로 SDK for Rust 작업을 호출하기 전에 main 함수 초기에 수행됩니다.

```
use aws_config::BehaviorVersion;

type BoxError = Box<dyn Error + Send + Sync>;

#[tokio::main]
async fn main() -> Result<(), BoxError> {
    tracing_subscriber::fmt::init(); // Initialize the subscriber.

    let config = aws_config::defaults(BehaviorVersion::latest())
        .load()
        .await;

    let s3 = aws_sdk_s3::Client::new(&config);

    let _resp = s3.list_buckets()
        .send()
        .await?;

    Ok(())
}
```

3. RUST\_LOG 환경 변수를 사용하여 로깅을 켭니다. 로깅 정보 표시를 활성화하려면 명령 프롬프트에서 RUST\_LOG 환경 변수를 로그인하려는 수준으로 설정합니다. 다음 예제에서는 로깅을 debug 수준으로 설정합니다.

## Linux/macOS

```
$ RUST_LOG=debug
```

## Windows

VSCode를 사용하는 경우 터미널 창은 주로 PowerShell로 기본 설정됩니다. 사용 중인 프롬프트 유형을 확인합니다.

```
C:\> set RUST_LOG=debug
```

## PowerShell

```
PS C:\> $ENV:RUST_LOG="debug"
```

### 4. 프로그램을 실행합니다:

```
$ cargo run
```

콘솔 또는 터미널 창에 추가 출력이 표시됩니다.

자세한 내용은 `tracing-subscriber` 설명서의 [환경 변수를 사용하여 이벤트 필터링](#)을 참조하세요.

## 로그 출력 해석

이전 섹션의 단계에 따라 로깅을 켜면 기본적으로 추가 로그 정보가 표준으로 인쇄됩니다.

기본 로그 출력 형식(추적 모듈에서 "전체"라고 함)을 사용하는 경우 로그 출력에 표시되는 정보는 다음과 같습니다.

```
2024-06-25T16:10:12.367482Z DEBUG invoke{service=s3 operation=ListBuckets
sdk_invocation_id=3434933}:try_op:try_attempt:lazy_load_identity:
aws_smithy_runtime::client::identity::cache::lazy: identity cache miss occurred;
added new identity (took 480.892ms) new_expiration=2024-06-25T23:07:59Z
valid_for=25066.632521s partition=IdentityCachePartition(7)
2024-06-25T16:10:12.367602Z DEBUG invoke{service=s3 operation=ListBuckets
sdk_invocation_id=3434933}:try_op:try_attempt:
aws_smithy_runtime::client::identity::cache::lazy: loaded identity
2024-06-25T16:10:12.367643Z TRACE invoke{service=s3 operation=ListBuckets
sdk_invocation_id=3434933}:try_op:try_attempt:
```

```
aws_smithy_runtime::client::orchestrator::auth: resolved identity identity=Identity
{ data: Credentials {... }, expiration: Some(SystemTime { tv_sec: 1719356879, tv_nsec:
0 }) }
2024-06-25T16:10:12.367695Z TRACE invoke{service=s3 operation=ListBuckets
sdk_invocation_id=3434933}:try_op:try_attempt:
aws_smithy_runtime::client::orchestrator::auth: signing request
```

각 항목에는 다음이 포함됩니다.

- 로그 항목의 타임스탬프입니다.
- 항목의 로그 수준입니다. 이는 INFO, DEBUG 또는 TRACE와 같은 단어입니다.
- 로그 항목이 생성된 중첩된 [범위](#) 집합으로, 콜론(":")으로 구분됩니다. 이렇게 하면 로그 항목의 소스를 식별할 수 있습니다.
- 로그 항목을 생성한 코드가 포함된 Rust 모듈 경로입니다.
- 로그 메시지 텍스트입니다.

추적 모듈의 표준 출력 형식은 ANSI 이스케이프 코드를 사용하여 출력을 색상화합니다. 출력을 필터링하거나 검색할 때 이러한 이스케이프 시퀀스를 염두에 두세요.

#### Note

중첩된 범위 집합 내에 `sdk_invocation_id` 나타나는 로그 메시지의 상관관계를 파악하는 데 도움이 되도록 SDK에서 생성한 고유한 ID 클라이언트 측입니다. 이는 응답에서 발견된 요청 ID와 관련이 없습니다 AWS 서비스.

## 로깅 수준 미세 조정

와 같은 환경 필터링을 지원하는 크레이트를 사용하는 경우 모듈별로 로그의 세부성을 제어할 `tracing_subscriber` 수 있습니다.

모든 모듈에 대해 동일한 로깅 수준을 켤 수 있습니다. 다음은 모든 모듈에 `trace` 대해 로깅 수준을 로 설정합니다.

```
$ RUST_LOG=trace cargo run
```

특정 모듈에 대해 트레이스 수준 로깅을 켤 수 있습니다. 다음 예제에서는에서 오는 로그만 `trace` 레벨로 `aws_smithy_runtime` 들어옵니다.



```
$ RUST_LOG=aws_smithy_runtime=trace
```

여러 모듈에 대해 쉼표로 구분하여 다른 로그 수준을 지정할 수 있습니다. 다음 예제에서는 `aws_config` 모듈을 `trace` 레벨 로깅으로 설정하고 `aws_smithy_runtime` 모듈을 `debug` 레벨 로깅으로 설정합니다.

```
$ RUST_LOG=aws_config=trace,aws_smithy_runtime=debug cargo run
```

다음 표에서는 로그 메시지를 필터링하는 데 사용할 수 있는 몇 가지 모듈을 설명합니다.

접두사	설명
<code>aws_smithy_runtime</code>	요청 및 응답 와이어 로깅
<code>aws_config</code>	자격 증명 로드
<code>aws_sigv4</code>	요청 서명 및 표준 요청

로그 출력에 포함해야 하는 모듈을 파악하는 한 가지 방법은 먼저 모든 것을 로그한 다음 필요한 정보를 위해 로그 출력에서 상자 이름을 찾는 것입니다. 그런 다음 그에 따라 환경 변수를 설정하고 프로그램을 다시 실행할 수 있습니다.

## 클라이언트의 단일 작업 구성 재정의

[서비스 클라이언트를 생성한](#) 후에는 구성을 변경할 수 없으며 모든 후속 작업에 적용됩니다. 이 시점에서 구성을 수정할 수 없지만 작업별로 재정의할 수 있습니다.

각 작업 빌더에는 기존 구성의 개별 복사본을 재정의할 수 `CustomizableOperation` 있도록 생성하는 데 사용할 수 있는 `customize` 메서드가 있습니다. 원래 클라이언트 구성은 수정되지 않은 상태로 유지됩니다.

다음 예제는 두 작업을 호출하는 Amazon S3 클라이언트의 생성을 보여줍니다. 두 번째 작업은 다른 로 전송하도록 재정의됩니다 AWS 리전. 모든 Amazon S3의 객체 호출은 수정된를 사용하기 위해 API 호출이 명시적으로 재정의되는 경우를 제외하고 `us-east-1` 리전을 사용합니다 `us-west-2`.

```
use aws_config::{BehaviorVersion, Region};

let config = aws_config::defaults(BehaviorVersion::latest())
```

```

    .region("us-east-1")
    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);

// Request will be sent to "us-east-1"
s3.list_buckets()
    .send()
    .await?;

// Unset fields default to using the original config value
let modified = aws_sdk_s3::Config::builder()
    .region(Region::from_static("us-west-2"));

// Request will be sent to "us-west-2"
s3.list_buckets()
    // Creates a CustomizableOperation
    .customize()
    .config_override(modified)
    .send()
    .await?;

```

### Note

이전 예제는 Amazon S3용이지만 개념은 모든 작업에 대해 동일합니다. 특정 작업에는에 대한 추가 메서드가 있을 수 있습니다CustomizableOperation.

단일 작업에 `customize`를 사용하여 인터셉터를 추가하는 예제는 섹션을 참조하세요 [특정 작업 전용 인터셉터](#).

## 재시도

는 기본 재시도 동작과 사용자 지정 가능한 구성 옵션을 AWS SDK for Rust 제공합니다. 예기치 않은 예외를 AWS 서비스 가끔 반환하도록 호출합니다. 통화를 재시도하면 제한 또는 일시적인 오류와 같은 특정 유형의 오류가 성공할 수 있습니다.

공유 AWS config 파일의 환경 변수 또는 설정을 사용하여 전역적으로 재시도 동작을 구성할 수 있습니다. 이 접근 방식에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 [재시도 동작](#)을 참조하세요. AWS SDKs 또한 재시도 전략 구현에 대한 자세한 정보와 하나를 선택하는 방법도 포함되어 있습니다.

또는 다음 섹션과 같이 코드에서 이러한 옵션을 구성할 수도 있습니다.

## 기본 재시도 구성

모든 서비스 클라이언트는 기본적으로 [RetryConfig](#) 구조를 통해 제공된 standard 재시도 전략 구성으로 설정됩니다. 기본적으로 호출은 세 번 시도됩니다(초기 시도와 두 번의 재시도). 또한 재시도 폭풍을 방지하기 위해 각 재시도가 짧은 무작위 기간만큼 지연됩니다. 이 규칙은 대부분의 사용 사례에 적합하지만 처리량이 많은 시스템과 같은 특정 상황에서는 적합하지 않을 수 있습니다.

일부 유형의 오류만 SDKs에서 재시도 가능한 것으로 간주됩니다. 재시도 가능한 오류의 예는 다음과 같습니다.

- 소켓 제한 시간
- 서비스 측 제한
- HTTP 5XX 응답과 같은 일시적인 서비스 오류

다음 예제는 재시도 가능한 것으로 간주되지 않습니다.

- 누락되거나 유효하지 않은 파라미터
- 인증/보안 오류
- 잘못된 구성 예외

최대 시도 횟수, 지연 시간 및 백오프 구성을 설정하여 standard 재시도 전략을 사용자 지정할 수 있습니다.

## 최대 시도 횟수

예 수정된를 제공하여 코드의 최대 시도 [RetryConfig](#) 횟수를 사용자 지정할 수 있습니다. `aws_config::defaults`.

```
const CUSTOM_MAX_ATTEMPTS: u32 = 5;
let retry_config = RetryConfig::standard()
    // Set max attempts. When max_attempts is 1, there are no retries.
    // This value MUST be greater than zero.
    // Defaults to 3.
    .with_max_attempts(CUSTOM_MAX_ATTEMPTS);
```

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .retry_config(retry_config)
    .load()
    .await;
```

## 지연 및 백오프

재시도가 필요한 경우 기본 재시도 전략은 후속 시도를 수행하기 전에 대기합니다. 첫 번째 재시도의 지연 시간은 작지만 이후 재시도에서는 기하급수적으로 증가합니다. 최대 지연 시간은 너무 크게 증가하지 않도록 제한됩니다.

무작위 지터는 모든 시도 사이의 지연에 적용됩니다. 지터는 재시도 폭풍을 일으킬 수 있는 대규모 플릿의 영향을 완화하는 데 도움이 됩니다. 지수 백오프 및 지터에 대한 자세한 내용은 AWS 아키텍처 블로그의 [지수 백오프 및 지터](#)를 참조하세요.

예 수정된를 제공하여 코드의 지연 설정을 사용자 지정할 수 [RetryConfig](#) 있습니다 `aws_config::defaults`. 다음 코드는 구성을 설정하여 최대 100밀리초 동안 첫 번째 재시도를 지연하고 재시도 사이의 최대 시간은 5초입니다.

```
let retry_config = RetryConfig::standard()
    // Defaults to 1 second.
    .with_initial_backoff(Duration::from_millis(100))
    // Defaults to 20 seconds.
    .with_max_backoff(Duration::from_secs(5));

let config = aws_config::defaults(BehaviorVersion::latest())
    .retry_config(retry_config)
    .load()
    .await;
```

## 적응형 재시도 모드

`standard` 모드 재시도 전략의 대안으로 `adaptive` 모드 재시도 전략은 제한 오류를 최소화하기 위한 이상적인 요청 속도를 찾는 고급 접근 방식입니다.

### Note

적응형 재시도는 고급 재시도 모드입니다. 이 전략을 사용하는 것은 일반적으로 권장되지 않습니다. SDK 및 도구 참조 안내서의 [재시도 동작](#)을 참조하세요. AWS SDKs

적응형 재시도에는 표준 재시도의 모든 기능이 포함됩니다. 제한되지 않은 요청과 비교하여 제한되는 요청의 속도를 측정하는 클라이언트 측 속도 제한기를 추가합니다. 또한 트래픽을 제한하여 안전한 대역폭 내에 유지하려고 시도하므로 제한 오류가 발생하지 않는 것이 이상적입니다.

속도는 변화하는 서비스 조건 및 트래픽 패턴에 실시간으로 적응하며 그에 따라 트래픽 속도를 높이거나 낮출 수 있습니다. 매우 심각하게, 속도 제한기는 트래픽이 많은 시나리오에서 초기 시도를 지연시킬 수 있습니다.

수정된를 제공하여 코드에서 adaptive 재시도 전략을 선택할 수 있습니다 [RetryConfig](#).

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .retry_config(RetryConfig::adaptive())
    .load()
    .await;
```

## 시간 초과

는 요청 제한 시간 및 중지된 데이터 스트림을 관리하기 위한 몇 가지 설정을 AWS SDK for Rust 제공합니다. 이를 통해 네트워크에서 예기치 않은 지연 및 장애가 발생할 때 애플리케이션이 최적으로 작동할 수 있습니다.

### API 제한 시간

요청 시도에 오랜 시간이 걸리거나 완전히 실패할 수 있는 일시적인 문제가 있는 경우 애플리케이션이 빠르게 실패하고 최적으로 작동할 수 있도록 제한 시간을 검토하고 설정하는 것이 중요합니다. 실패한 요청은 SDK에서 자동으로 재시도할 수 있습니다. 개별 시도와 전체 요청 모두에 대해 제한 시간을 설정하는 것이 좋습니다.

SDK for Rust는 요청에 대한 연결을 설정하기 위한 기본 제한 시간을 제공합니다. SDK에는 요청 시도 또는 전체 요청에 대한 응답을 수신하기 위한 기본 최대 대기 시간이 설정되어 있지 않습니다. 다음 제한 시간 옵션을 사용할 수 있습니다.

파라미터	기본값	설명
연결 제한 시간	3.1초	포기하기 전에 연결을 설정하기 위해 대기하는 최대 시간입니다.
작업 제한 시간	없음	모든 재시도를 포함하여 SDK for Rust로부터 응답을 받기 전에 대기하는 최대 시간입니다.

파라미터	기본값	설명
작업 시도 제한 시간	없음	단일 HTTP 시도를 기다리는 최대 시간이며, 그 이후에는 API 호출을 재시도할 수 있습니다.
읽기 제한 시간	없음	요청이 시작된 시점부터 응답의 첫 번째 바이트를 읽기 위해 대기하는 최대 시간입니다.

다음 예제는 사용자 지정 제한 시간 값이 있는 Amazon S3 클라이언트의 구성을 보여줍니다.

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .timeout_config(
        TimeoutConfig::builder()
            .operation_timeout(Duration::from_secs(5))
            .operation_attempt_timeout(Duration::from_millis(1500))
            .build()
    )
    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);
```

작업과 시도 제한 시간을 모두 함께 사용하는 경우 재시도의 모든 시도에 소요된 총 시간에 대해 하드 제한을 설정합니다. 또한 느린 요청에 대해 빠르게 실패하도록 개별 HTTP 요청을 설정합니다.

모든 작업에 대해 서비스 클라이언트에서 이러한 제한 시간 값을 설정하는 대신 [단일 요청에 대해 이러한 제한 시간 값을 구성하거나 재정의](#)할 수 있습니다.

#### Important

SDK for Rust가 응답을 반환한 후 사용된 스트리밍 데이터에는 작업 및 시도 제한 시간이 적용되지 않습니다. 예를 들어 응답의 `ByteStream` 멤버에서 데이터를 사용하는 경우 작업 제한 시간이 적용되지 않습니다.

## 스톱 스트림 보호

SDK for Rust는 중지된 스트림 감지와 관련된 또 다른 형태의 제한 시간을 제공합니다. 중지된 스트림은 구성된 유예 기간보다 오래 데이터를 생성하지 않는 업로드 또는 다운로드 스트림입니다. 이렇게 하면 애플리케이션이 무기한 중단되고 진행되지 않도록 방지할 수 있습니다.

스트림이 허용 기간보다 오래 유휴 상태인 경우 스트림 보호가 중단되면 오류가 반환됩니다.

기본적으로 SDK for Rust는 업로드 및 다운로드 모두에 대해 스트림 차단을 활성화하고 20초의 충분한 유예 기간으로 최소 1바이트/초의 활동을 찾습니다.

다음 예제에서는 업로드 보호를 `StalledStreamProtectionConfig` 비활성화하고 활동 없는 유예 기간을 10초로 변경하는 사용자 지정을 보여줍니다.

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .stalled_stream_protection(
        StalledStreamProtectionConfig::enabled()
            .upload_enabled(false)
            .grace_period(Duration::from_secs(10))
            .build()
    )
    .load()
    .await;
```

### Warning

중단된 스트림 보호는 고급 구성 옵션입니다. 애플리케이션에 더 엄격한 성능이 필요하거나 다른 문제가 발생하는 경우에만 이러한 값을 변경하는 것이 좋습니다.

## 중지된 스트림 보호 비활성화

다음 예제에서는 지연된 스트림 보호를 완전히 비활성화하는 방법을 보여줍니다.

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .stalled_stream_protection(StalledStreamProtectionConfig::disabled())
    .load()
    .await;
```

# 사용 AWS SDK for Rust

를 사용하여 AWS 서비스를 AWS SDK for Rust 사용하는 일반적인 방법과 권장되는 방법을 알아봅니다.

## 주제

- [요청을 생성](#)
- [모범 사례](#)
- [동시성](#)
- [오류 처리](#)
- [Lambda 함수 생성](#)
- [페이지 매김](#)
- [미리 서명된 URLs 생성](#)
- [유닛 테스트](#)
- [Waiters](#)

## 요청을 생성

에 요청하려면 먼저 [서비스 클라이언트를 생성](#) AWS 서비스해야 합니다. 가 사용하는 각 AWS 서비스 코드에는 자체 크레딧과 상호 작용Client용 자체 크레딧이 있습니다.

는 서비스에서 노출되는 각 API 작업에 대해 하나의 메서드를 Client 노출합니다. 이러한 각 메서드의 반환 값은 'fluent builder'이며, 빌더 스타일 함수 호출 체인에 의해 해당 API에 대한 다양한 입력이 추가됩니다. 서비스의 메서드를 호출한 후를 호출send()하여를 가져오Future면 출력이 성공하거나 가 됩니다SdkError. SdkError에 대한 자세한 내용은 [오류 처리](#) 섹션을 참조하세요.

다음 예제에서는 Amazon S3를 사용하여에서 버킷을 생성하는 기본 작업을 보여줍니다us-west-2 AWS 리전.

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);

let result = s3.create_bucket()
```



```
// Set some of the inputs for the operation.
.bucket("my-bucket")
.create_bucket_configuration(
    CreateBucketConfiguration::builder()
        .location_constraint(aws_sdk_s3::types::BucketLocationConstraint::UsWest2)
        .build()
)
// send() returns a Future that does nothing until awaited.
.send()
.await;
```

각 서비스 크레이트에는 다음과 같이 API 입력에 사용되는 추가 모듈이 있습니다.

- `types` 모듈에는 보다 복잡한 구조화된 정보를 제공하는 구조 또는 열거형이 있습니다.
- `primitives` 모듈에는 날짜 시간 또는 바이너리 BLOB과 같은 데이터를 나타내는 더 간단한 유형이 있습니다.

자세한 상자 구성 및 정보는 서비스 상자의 [API 참조 설명서를](#) 참조하세요. 예를 들어 Amazon Simple Storage Service의 `aws-sdk-s3` 크레이트에는 여러 [모듈이 있습니다](#). 그 중 두 가지는 다음과 같습니다.

- [aws\\_sdk\\_s3::types](#)
- [aws\\_sdk\\_s3::primitives](#)

## 모범 사례

다음은 사용 모범 사례입니다 AWS SDK for Rust.

### 가능한 경우 SDK 클라이언트 재사용

SDK 클라이언트 구성 방식에 따라 새 클라이언트를 생성하면 각 클라이언트가 자체 HTTP 연결 풀, 자격 증명 캐시 등을 유지 관리할 수 있습니다. 값비싼 리소스 생성으로 인한 오버헤드를 방지하려면 클라이언트를 공유하거나 최소한 공유하는 `SdkConfig` 것이 좋습니다. 모든 SDK 클라이언트는 단일 원자 참조 수 업데이트 Clone로 구현됩니다.

### API 타임아웃 설정

SDK는 연결 제한 시간 및 소켓 제한 시간과 같은 일부 제한 시간 옵션에 대한 기본값을 제공하지만 API 호출 제한 시간 또는 개별 API 호출 시도에는 기본값을 제공하지 않습니다. 개별 시도와 전체 요청

모두에 대해 제한 시간을 설정하는 것이 좋습니다. 이렇게 하면 요청 시도를 완료하는 데 시간이 더 오래 걸리거나 치명적인 네트워크 문제를 일으킬 수 있는 일시적인 문제가 있는 경우 최적의 방식으로 애플리케이션이 빠르게 실패하도록 보장합니다.

작업 제한 시간 구성에 대한 자세한 내용은 섹션을 참조하세요 [시간 초과](#).

## 동시성

AWS SDK for Rust 는 동시성 제어를 제공하지 않지만 사용자에게는 자체 구현을 위한 많은 옵션이 있습니다.

## 용어

이 주제와 관련된 용어는 혼동하기 쉬우며, 일부 용어는 원래 별도의 개념을 표현했다라도 동의어가 되었습니다. 이 가이드에서는 다음을 정의합니다.

- **작업:** 프로그램이 완료될 때까지 실행되거나 완료될 때까지 실행하려고 시도하는 일부 "작업 단위"입니다.
- **순차 컴퓨팅:** 여러 작업이 차례로 실행되는 경우.
- **동시 컴퓨팅:** 여러 태스크가 중첩 기간에 실행되는 경우.
- **동시성:** 컴퓨터가 임의의 순서로 여러 작업을 완료하는 기능입니다.
- **멀티태스킹:** 컴퓨터가 여러 작업을 동시에 실행하는 기능입니다.
- **레이스 조건:** 작업이 시작된 시간 또는 작업을 처리하는 데 걸리는 시간에 따라 프로그램의 동작이 변경되는 경우.
- **경합:** 공유 리소스에 대한 액세스와 충돌합니다. 둘 이상의 태스크가 리소스에 동시에 액세스하려는 경우 해당 리소스는 "경합"입니다.
- **잠금 해제:** 더 이상 진행할 수 없는 상태입니다. 이는 일반적으로 두 태스크가 서로의 리소스를 획득하려고 하지만 두 태스크 모두 다른 태스크의 리소스를 사용할 수 있을 때까지 리소스를 해제하지 않기 때문에 발생합니다. 잠금이 해제되면 프로그램이 부분적으로 또는 완전히 응답하지 않게 됩니다.

## 간단한 예

첫 번째 예제는 순차적 프로그램입니다. 이후 예제에서는 동시성 기술을 사용하여이 코드를 변경합니다. 이후 예제에서는 동일한 `build_client_and_list_objects_to_download()` 메서드를 재사용하고 내에서 변경합니다 `main()`.

다음 예제 작업은 Amazon Simple Storage Service 버킷의 모든 파일을 다운로드하는 것입니다.

1. 먼저 모든 파일을 나열합니다. 목록에 키를 저장합니다.
2. 목록을 반복하여 각 파일을 차례로 다운로드합니다.

```
const EXAMPLE_BUCKET: &str = "<an-example-bucket>";

// This initialization function won't be reproduced in
// examples following this one, in order to save space.
async fn build_client_and_list_objects_to_download() {
    let cfg = aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
    let client = Client::new(&cfg);
    let objects_to_download: Vec<_> = client
        .list_objects_v2()
        .bucket(EXAMPLE_BUCKET)
        .send()
        .await
        .expect("listing objects succeeds")
        .contents()
        .into_iter()
        .flat_map(aws_sdk_s3::types::Object::key)
        .map(ToString::to_string)
        .collect();

    (client, objects_to_download)
}
```

```
#[tokio::main]
async fn main() {
    let (client, objects_to_download) =
        build_client_and_list_objects_to_download().await;

    for object in objects_to_download {
        let res = client
            .get_object()
            .key(&object)
            .bucket(EXAMPLE_BUCKET)
            .send()
            .await
            .expect("get_object succeeds");
        let body = res.body.collect().await.expect("reading body
succeeds").into_bytes();
        std::fs::write(object, body).expect("write succeeds");
    }
}
```

```
}
}
```

### Note

이 예제에서는 오류를 처리하지 않으며 예제 버킷에 파일 경로와 같은 키가 있는 객체가 없다고 가정합니다. 따라서 중첩 디렉터리 생성은 다루지 않습니다.

최신 컴퓨터의 아키텍처로 인해서 프로그램을 훨씬 더 효율적으로 다시 작성할 수 있습니다. 이후 예제에서는 이 작업을 수행하지만 먼저 몇 가지 개념을 더 살펴보겠습니다.

## 소유권 및 변경 가능성

Rust의 각 값에는 단일 소유자가 있습니다. 소유자가 범위를 벗어나면 소유한 모든 값도 삭제됩니다. 소유자는 값에 대한 하나 이상의 변경 불가능한 참조 또는 단일 변경 가능한 참조를 제공할 수 있습니다. Rust 컴파일러는 참조가 소유자보다 오래 지속되지 않도록 할 책임이 있습니다.

여러 태스크가 동일한 리소스에 변경 가능하게 액세스해야 하는 경우 추가 계획 및 설계가 필요합니다. 순차 컴퓨팅에서는 각 태스크가 시퀀스에서 차례로 실행되므로 경합 없이 동일한 리소스에 변경 가능하게 액세스할 수 있습니다. 그러나 동시 컴퓨팅에서는 태스크가 어떤 순서로든 동시에 실행될 수 있습니다. 따라서 여러 개의 변경 가능한 참조가 불가능하다는 것을 컴파일러에 증명하기 위해(또는 최소한 발생할 경우 충돌을 위해) 더 많은 작업을 수행해야 합니다.

Rust 표준 라이브러리는 이를 달성하는 데 도움이 되는 다양한 도구를 제공합니다. 이러한 주제에 대한 자세한 내용은 Rust 프로그래밍 언어 책의 [변수 및 변경 가능성](#)과 [소유권 이해](#)를 참조하세요.

## 추가 용어!

다음은 "동기화 객체"의 목록입니다. 또한 동시 프로그램이 소유권 규칙을 위반하지 않을 것이라고 컴파일러를 설득하는 데 필요한 도구입니다.

### 표준 라이브러리 동기화 객체:

- [아크](#): Atomically Reference-Counted 포인터입니다. 데이터가 래핑되면 특정 소유자가 값을 조기에 삭제할 걱정 없이 데이터를 자유롭게 공유할 Arc 수 있습니다. 이러한 의미에서 값의 소유권은 "공유"가 됩니다. 내 값은 변경할 Arc 수 없지만 [내부는 변경할](#) 수 있습니다.
- [장벽](#): 여러 스레드가 프로그램의 특정 지점에 도달할 때까지 기다린 후 모두 계속 실행합니다.

- [Condvar](#): 이벤트가 발생할 때까지 기다리는 동안 스레드를 차단하는 기능을 제공하는 Condition Variable입니다.
- [Mutex](#): 한 번에 최대 하나의 스레드가 일부 데이터에 액세스할 수 있도록 하는 Mutual Ex 복제 메커니즘입니다. 일반적으로 Mutex 잠금은 코드의 한 `.await` 지점에 걸쳐 유지해서는 안 됩니다.

### [Tokio 동기화 객체:](#)

AWS SDKs `async-runtime-agnostic`으로 설계되었지만 특정 경우에 `tokio` 동기화 객체를 사용하는 것이 좋습니다.

- [Mutex](#): 표준 라이브러리의와 비슷하지만 비용이 약간 더 높습니다. 표준과 달리 `Mutex`이는 코드의 한 `.await` 지점에 걸쳐 보유할 수 있습니다.
- [Semaphore](#): 여러 태스크로 공통 리소스에 대한 액세스를 제어하는 데 사용되는 변수입니다.

## 더 효율적으로 예제를 다시 작성(단일 스레드 동시성)

다음 수정된 예제에서는 [futures\\_util::future::join\\_all](#)를 사용하여 ALL `get_object` 요청을 동시에 실행합니다.

```
#[tokio::main]
async fn main() {
    let (client, objects_to_download) =
        build_client_and_list_objects_to_download().await;

    let get_object_futures = objects_to_download.into_iter().map(|object| {
        let req = client
            .get_object()
            .key(&object)
            .bucket(EXAMPLE_BUCKET);

        async {
            let res = req
                .send()
                .await
                .expect("get_object succeeds");
            let body = res.body.collect().await.expect("body succeeds").into_bytes();
            // Note that we MUST use the async runtime's preferred way
            // of writing files. Otherwise, this call would block,
            // potentially causing a deadlock.
            tokio::fs::write(object, body).await.expect("write succeeds");
        }
    });
}
```

```

    }
  });

  futures_util::future::join_all(get_object_futures).await;
}

```

이는 동시성의 이점을 가장 간단하게 얻을 수 있는 방법이지만, 언뜻 보기에는 명확하지 않을 수 있는 몇 가지 문제도 있습니다.

1. 모든 요청 입력을 동시에 생성합니다. 모든 `get_object` 요청 입력을 보관할 메모리가 충분하지 않은 경우 "out-of-memory" 할당 오류가 발생합니다.
2. 동시에 모든 선물을 생성하고 기다립니다. Amazon S3는 한 번에 너무 많이 다운로드하려고 하면 요청을 제한합니다.

이 두 가지 문제를 모두 해결하려면 한 번에 보내는 요청의 양을 제한해야 합니다. 세 `tokio` [마포어](#)를 사용하여 이 작업을 수행합니다.

```

const CONCURRENCY_LIMIT: usize = 50;

#[tokio::main(flavor = "current_thread")]
async fn main() {
    let (client, objects_to_download) =
        build_client_and_list_objects_to_download().await;
    let concurrency_semaphore = Arc::new(Semaphore::new(CONCURRENCY_LIMIT));

    let get_object_futures = objects_to_download.into_iter().map(|object| {
        // Since each future needs to acquire a permit, we need to clone
        // the Arc'd semaphore before passing it in.
        let semaphore = concurrency_semaphore.clone();
        // We also need to clone the client so each task has its own handle.
        let client = client.clone();
        async move {
            let permit = semaphore
                .acquire()
                .await
                .expect("we'll get a permit if we wait long enough");
            let res = client
                .get_object()
                .key(&object)
                .bucket(EXAMPLE_BUCKET)
                .send()

```

```

        .await
        .expect("get_object succeeds");
    let body = res.body.collect().await.expect("body succeeds").into_bytes();
    tokio::fs::write(object, body).await.expect("write succeeds");
    std::mem::drop(permit);
    }
});

futures_util::future::join_all(get_object_futures).await;
}

```

요청 생성을 `async` 블록으로 이동하여 잠재적인 메모리 사용량 문제를 해결했습니다. 이렇게 하면 요청을 보낼 시간이 될 때까지 요청이 생성되지 않습니다.

### Note

메모리가 있는 경우 모든 요청 입력을 한 번에 생성하고 전송할 준비가 될 때까지 메모리에 보관하는 것이 더 효율적일 수 있습니다. 이렇게 하려면 `async` 블록 외부로 요청 입력 생성을 이동합니다.

또한 비행 중인 요청을 로 제한하여 한 번에 너무 많은 요청을 보내는 문제를 해결했습니다. `CONCURRENCY_LIMIT`.

### Note

의 올바른 값은 모든 프로젝트에 따라 `CONCURRENCY_LIMIT` 다릅니다. 자체 요청을 구성하고 전송할 때 제한 오류 없이 최대한 높게 설정해 보세요. 서비스가 반응하는 응답의 성공률과 제한된 응답의 비율을 기반으로 동시성 제한을 동적으로 업데이트할 수 있지만 복잡성으로 인해 이 가이드의 범위를 벗어납니다.

## 더 효율적으로 예제를 다시 작성(다중 스레드 동시성)

이전 두 예제에서는 요청을 동시에 수행했습니다. 이는 동기식으로 실행하는 것보다 효율적이지만 멀티스레딩을 사용하여 사물을 더 효율적으로 만들 수 있습니다. 에서 이를 수행하려면 별도의 작업으로 생성 `tokio`해야 합니다.

**Note**

이 예제에서는 다중 스레드 tokio 런타임을 사용해야 합니다. 이 런타임은 `rt-multi-thread` 기능 뒤에 게이트됩니다. 물론 멀티 코어 시스템에서 프로그램을 실행해야 합니다.

```
// Set this based on the amount of cores your target machine has.
const THREADS: usize = 8;

#[tokio::main(flavor = "multi_thread")]
async fn main() {
    let (client, objects_to_download) =
        build_client_and_list_objects_to_download().await;
    let concurrency_semaphore = Arc::new(Semaphore::new(THREADS));

    let get_object_task_handles = objects_to_download.into_iter().map(|object| {
        // Since each future needs to acquire a permit, we need to clone
        // the Arc'd semaphore before passing it in.
        let semaphore = concurrency_semaphore.clone();
        // We also need to clone the client so each task has its own handle.
        let client = client.clone();

        // Note this difference! We're using `tokio::task::spawn` to
        // immediately begin running these requests.
        tokio::task::spawn(async move {
            let permit = semaphore
                .acquire()
                .await
                .expect("we'll get a permit if we wait long enough");
            let res = client
                .get_object()
                .key(&object)
                .bucket(EXAMPLE_BUCKET)
                .send()
                .await
                .expect("get_object succeeds");
            let body = res.body.collect().await.expect("body succeeds").into_bytes();
            tokio::fs::write(object, body).await.expect("write succeeds");
            std::mem::drop(permit);
        })
    });
};
```



```
futures_util::future::join_all(get_object_task_handles).await;
}
```

작업을 작업으로 나누는 것은 복잡할 수 있습니다. I/O(입력/출력)를 수행하는 것은 일반적으로 차단됩니다. 런타임은 장기 실행 작업의 요구 사항과 단기 실행 작업의 요구 사항 간의 균형을 맞추는 데 어려움을 겪을 수 있습니다. 어떤 런타임을 선택하든 작업을 작업으로 나누는 가장 효율적인 방법을 위해 권장 사항을 읽어야 합니다. tokio 런타임 권장 사항은 [모듈 섹션을 tokio::task](#) 참조하세요.

## 다중 스레드 앱 디버깅

동시에 실행되는 작업은 어떤 순서로든 실행할 수 있습니다. 따라서 동시 프로그램의 로그를 읽기가 매우 어려울 수 있습니다. SDK for Rust에서는 tracing 로깅 시스템을 사용하는 것이 좋습니다. 실행 시기와 관계없이 로그를 특정 작업과 그룹화할 수 있습니다. 자세한 지침은 [AWS SDK for Rust 코드 로깅 활성화](#)를 참조하세요.

잠긴 작업을 식별하는 데 매우 유용한 도구는 비동기 Rust 프로그램을 위한 진단 및 디버깅 도구 [tokio-console](#)인입니다. 프로그램을 구성하고 실행한 다음 tokio-console 앱을 실행하면 프로그램이 실행 중인 작업을 실시간으로 볼 수 있습니다. 이 보기에는 작업이 공유 리소스 획득을 기다리는 데 소요된 시간 또는 폴링된 시간과 같은 유용한 정보가 포함되어 있습니다.

## 오류 처리

SDK를 사용하여 고품질 애플리케이션을 구축하려면 오류를 AWS SDK for Rust 반환하는 방법과 시기를 이해하는 것이 중요합니다. 다음 섹션에서는 SDK에서 발생할 수 있는 다양한 오류와 이를 적절하게 처리하는 방법을 설명합니다.

모든 작업은 오류 Result 유형이 로 설정된 유형을 반환합니다 [SdkError<E, R = HttpResponse>](#). SdkError는 변형이라고 하는 몇 가지 가능한 유형이 있는 열거형입니다.

## 서비스 오류

가장 일반적인 유형의 오류는 [SdkError::ServiceError](#)입니다. 이 오류는의 오류 응답을 나타냅니다 AWS 서비스. 예를 들어 Amazon S3에서 존재하지 않는 객체를 가져오려고 하면 Amazon S3는 오류 응답을 반환합니다.

가 발생하면 요청이 로 성공적으로 전송되었지만 처리할 AWS 서비스 수 없음을 [SdkError::ServiceError](#) 의미합니다. 이는 요청의 파라미터 오류 또는 서비스 측의 문제로 인해 발생할 수 있습니다.

오류 응답 세부 정보는 오류 변형에 포함됩니다. 다음 예제에서는 기본 `ServiceError` 변형을 편리하게 가져오고 다양한 오류 사례를 처리하는 방법을 보여줍니다.

```
// Needed to access the '.code()' function on the error type:
use aws_sdk_s3::error::ProvideErrorMetadata;

let result = s3.get_object()
    .bucket("my-bucket")
    .key("my-key")
    .send()
    .await;

match result {
    Ok(_output) => { /* Success. Do something with the output. */ }
    Err(err) => match err.into_service_error() {
        GetObjectError::InvalidObjectState(value) => {
            println!("invalid object state: {:?}", value);
        }
        GetObjectError::NoSuchKey(_) => {
            println!("object didn't exist");
        }
        // err.code() returns the raw error code from the service and can be
        // used as a last resort for handling unmodeled service errors.
        err if err.code() == Some("SomeUnmodeledError") => {}
        err => return Err(err.into())
    }
};
```

## 오류 메타데이터

모든 서비스 오류에는 서비스별 특성을 가져와서 액세스할 수 있는 추가 메타데이터가 있습니다.

- `<service>::error::ProvideErrorMetadata` 특성은 서비스에서 반환된 사용 가능한 기본 원시 오류 코드 및 오류 메시지에 대한 액세스를 제공합니다.
  - Amazon S3의 경우이 특성은 [aws\\_sdk\\_s3::error::ProvideErrorMetadata](#).

서비스 오류 문제를 해결할 때 유용할 수 있는 정보를 얻을 수도 있습니다.

- `<service>::operation::RequestId` 특성은 확장 메서드를 추가하여 서비스에서 생성된 고유한 AWS 요청 ID를 검색합니다.
  - Amazon S3의 경우이 특성은 [aws\\_sdk\\_s3::operation::RequestId](#).

- `<service>::operation::RequestIdExt` 특성은 `extended_request_id()` 메서드를 추가하여 추가 확장 요청 ID를 가져옵니다.
- 일부 서비스에서만 지원됩니다.
- Amazon S3의 경우이 특성은 [입니디아 `aws\_sdk\_s3::operation::RequestIdExt`](#).

## 를 사용하여 세부 오류 인쇄 `DisplayErrorContext`

SDK의 오류는 일반적으로 다음과 같은 장애 체인의 결과입니다.

1. 커넥터가 오류를 반환했기 때문에 요청을 디스패치하지 못했습니다.
2. 자격 증명 공급자가 오류를 반환했기 때문에 커넥터가 오류를 반환했습니다.
3. 자격 증명 공급자가 서비스를 호출하고 해당 서비스가 오류를 반환했기 때문에 오류를 반환했습니다.
4. 보안 인증 요청에 올바른 권한이 없어 서비스가 오류를 반환했습니다.

기본적으로이 오류를 표시하면 "디스패치 실패"만 출력됩니다. 여기에는 오류를 해결하는데 도움이 되는 세부 정보가 없습니다. SDK for Rust는 라는 간단한 오류 리포터를 제공합니다 `DisplayErrorContext`.

- `<service>::error::DisplayErrorContext` 구조체는 전체 오류 컨텍스트를 출력하는 기능을 추가합니다.
- Amazon S3의 경우이 구조는 [입니디아 `aws\_sdk\_s3::error::DisplayErrorContext`](#).

표시할 오류를 래핑하고 인쇄할 때는 다음과 유사한 훨씬 더 자세한 메시지를 `DisplayErrorContext` 제공합니다.

```
dispatch failure: other: Session token not found or invalid.
DispatchFailure(
  DispatchFailure {
    source: ConnectorError {
      kind: Other(None),
      source: ProviderError(
        ProviderError {
          source: ProviderError(
            ProviderError {
              source: ServiceError(
                ServiceError {
```

```

        source: UnauthorizedException(
            UnauthorizedException {
                message: Some("Session token not found or
invalid"),
                meta: ErrorMetadata {
                    code: Some("UnauthorizedException"),
                    message: Some("Session token not found
or invalid"),
                    extras: Some({"aws_request_id":
"1b6d7476-f5ec-4a16-9890-7684ccee7d01"})
                }
            },
            raw: Response {
                status: StatusCode(401),
                headers: Headers {
                    headers: {
                        "date": HeaderValue { _private:
H0("Thu, 04 Jul 2024 07:41:21 GMT") },
                        "content-type": HeaderValue { _private:
H0("application/json") },
                        "content-length": HeaderValue
{ _private: H0("114") },
                        "access-control-expose-headers":
HeaderValue { _private: H0("RequestId") },
                        "access-control-expose-headers":
HeaderValue { _private: H0("x-amzn-RequestId") },
                        "requestid": HeaderValue { _private:
H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") },
                        "server": HeaderValue { _private:
H0("AWS SSO") },
                        "x-amzn-requestid": HeaderValue
{ _private: H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") }
                    }
                },
                body: SdkBody {
                    inner: Once(
                        Some(
                            b"{
                                \"message\": \"Session token not
found or invalid\",
                                \"__type\":
\"com.amazonaws.switchboard.portal#UnauthorizedException\"}"
                        )
                    )
                }
            }
        )
    }
}

```

```

        ),
        retryable: true
    },
    extensions: Extensions {
        extensions_02x: Extensions,
        extensions_1x: Extensions
    }
}
)
}
),
connection: Unknown
}
}
)

```

## Lambda 함수 생성

를 사용하여 AWS Lambda 함수를 개발하는 방법에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 [Rust를 사용하여 Lambda 함수 빌드](#)를 AWS SDK for Rust 참조하세요. 이 설명서는 다음을 사용하는 방법을 안내합니다.

- 코어 기능에 대한 Rust Lambda 런타임 클라이언트 크레딧입니다 [aws-lambda-rust-runtime](#).
- [Cargo Lambda](#)를 사용하여 Rust 함수 바이너리를 Lambda에 배포하기 위한 권장 명령줄 도구입니다.

AWS Lambda 개발자 안내서의 안내 예제 외에도 GitHub의 [AWS SDK 코드 예제 리포지토리](#)에서 Lambda 계산기 예제를 사용할 수 있습니다.

## 페이지 매김

페이로드가 너무 커서 단일 응답으로 반환할 수 없는 경우 많은 AWS 작업이 잘린 결과를 반환합니다. 대신 서비스는 데이터의 일부와 토큰을 반환하여 다음 항목 세트를 검색합니다. 이 패턴을 페이지 매김이라고 합니다.

에는 결과를 자동으로 페이지 매김하는 데 사용할 수 있는 `into_paginator` 작업 빌더의 확장 메서드가 AWS SDK for Rust 포함되어 있습니다. 결과를 처리하는 코드만 작성하면 됩니다. 모

든 페이지 매김 작업 빌더에는 `노출 PaginationStream<Item>하여 결과를 페이지 매김하는 into_paginator() 메서드가 있습니다.`

- Amazon S3에서 이에 대한 한 가지 예는 입니

다 [aws\\_sdk\\_s3::operation::list\\_objects\\_v2::builders::ListObjectsV2FluentBuilder:](#)

다음 예제에서는 Amazon Simple Storage Service를 사용합니다. 그러나 하나 이상의 페이지가 지정된 APIs가 있는 모든 서비스에서 개념은 동일합니다.

다음 코드 예제는 `try_collect()` 메서드를 사용하여 페이지가 매겨진 모든 결과를 로 수집하는 가장 간단한 예제를 보여줍니다. `Vec`

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);

let all_objects = s3.list_objects_v2()
    .bucket("my-bucket")
    .into_paginator()
    .send()
    .try_collect()
    .await?
    .into_iter()
    .flat_map(|o| o.contents.unwrap_or_default())
    .collect::<Vec<_>>();
```

페이징을 더 잘 제어하고 모든 것을 한 번에 메모리로 가져오지 않으려는 경우가 있습니다. 다음 예제에서는 더 이상 없을 때까지 Amazon S3 버킷의 객체를 반복합니다.

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);

let mut paginator = s3.list_objects_v2()
    .bucket("my-bucket")
    .into_paginator()
    // customize the page size (max results per/response)
```

```

    .page_size(10)
    .send();

println!("Objects in bucket:");

while let Some(result) = paginator.next().await {
    let resp = result?;
    for obj in resp.contents() {
        println!("\t{:?}", obj);
    }
}
}

```

## 미리 서명된 URLs 생성

나중에 다른 호출자가 자신의 자격 증명을 제시하지 않고 요청을 사용할 수 있도록 일부 AWS API 작업에 대한 요청에 미리 서명할 수 있습니다.

예를 들어 Jane이 Amazon Simple Storage Service(Amazon S3) 객체에 액세스할 수 있고 Alejandro와 객체 액세스를 일시적으로 공유하려고 한다고 가정해 보겠습니다. Jane은 미리 서명된 GetObject 요청을 생성하여 Alejandro와 공유하므로 Jane의 자격 증명에 액세스하거나 자신의 자격 증명을 보유하지 않고도 객체를 다운로드할 수 있습니다. 미리 서명된 URL에서 사용하는 자격 증명은 Jane의 자격 증명입니다. Jane은 URL을 생성한 AWS 사용자이기 때문입니다.

Amazon S3의 미리 서명된 URLs에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [미리 서명된 URLs 작업을 참조하세요](#).

## 기본 사항 사전 지정

는 미리 서명된 요청을 가져오는 데 사용할 수 있는 작업 유창 빌더에 대한 `presigned()` 메서드를 AWS SDK for Rust 제공합니다.

다음 예제에서는 Amazon S3에 대해 미리 서명된 GetObject 요청을 생성합니다. 요청은 생성 후 5분 동안 유효합니다.

```

use std::time::Duration;
use aws_config::BehaviorVersion;
use aws_sdk_s3::presigning::PresigningConfig;

let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

```

```
let s3 = aws_sdk_s3::Client::new(&config);

let presigned = s3.get_object()
    .presigned(
        PresigningConfig::builder()
            .expires_in(Duration::from_secs(60 * 5))
            .build()
            .expect("less than one week")
    )
    .await?;
```

`presigned()` 메서드는 `Result<PresignedRequest, SdkError<E, R>>`를 반환합니다.

반환된 `PresignedRequest`에는 메서드, URI 및 헤더를 포함하여 HTTP 요청의 구성 요소를 가져오는 메서드가 포함되어 있습니다. 요청이 유효하려면 이러한 모든 항목을 서비스로 전송해야 합니다. 그러나 미리 서명된 많은 요청은 URI만으로 표현할 수 있습니다.

## POST 및 PUT 요청 사전 서명

미리 서명할 수 있는 많은 작업에는 URL만 필요하며 HTTP GET 요청으로 전송해야 합니다. 그러나 일부 작업은 본문을 사용하며 경우에 따라 헤더와 함께 HTTP POST 또는 HTTP PUT 요청으로 전송해야 합니다. 이러한 요청에 미리 서명하는 것은 GET 요청에 미리 서명하는 것과 동일하지만 미리 서명된 요청을 호출하는 것은 더 복잡합니다.

다음은 Amazon S3 `PutObject` 요청을 미리 서명하고 선택한 HTTP 클라이언트를 사용하여 전송할 수 있는 `Request` 로 변환하는 예제입니다.

`into_http_1x_request()` 메서드를 사용하려면 `Cargo.toml` 파일의 `aws-sdk-s3` 크레이트에 `http-1x` 기능을 추가합니다.

```
aws-sdk-s3 = { version = "1", features = ["http-1x"] }
```

소스 파일:

```
let presigned = s3.put_object()
    .presigned(
        PresigningConfig::builder()
            .expires_in(Duration::from_secs(60 * 5))
            .build()
            .expect("less than one week")
    )
    .await?;
```



```

)
.await?;

let body = "Hello AWS SDK for Rust";
let http_req = presigned.into_http_1x_request(body);

```

## 독립 실행형 서명자

### Note

이는 고급 사용 사례입니다. 대부분의 사용자에게 필요하거나 권장되지는 않습니다.

SDK for Rust 컨텍스트 외부에서 서명된 요청을 생성해야 하는 몇 가지 사용 사례가 있습니다. 의 경우 SDK와 독립적으로 [aws-sigv4](#) 크레이트를 사용할 수 있습니다.

다음은 기본 요소를 보여주는 예제입니다. 자세한 내용은 상자 설명서를 참조하세요.

Cargo.toml 파일에 aws-sigv4 및 http 상자를 추가합니다.

```

[dependencies]
aws-sigv4 = "1"
http = "1"

```

### 소스 파일:

```

use aws_smithy_runtime_api::client::identity::Identity;
use aws_sigv4::http_request::{sign, SigningSettings, SigningParams, SignableRequest};
use aws_sigv4::sign::v4;
use std::time::SystemTime;

// Set up information and settings for the signing.
// You can obtain credentials from `SdkConfig`.
let identity = Credentials::new(
    "AKIDEXAMPLE",
    "wJalrXUtnFEMI/K7MDENG+bPxrFiCYEXAMPLEKEY",
    None,
    None,
    "hardcoded-credentials").into();

```

```

let settings = SigningSettings::default();

let params = v4::SigningParams::builder()
    .identity(&identity)
    .region("us-east-1")
    .name("service")
    .time(SystemTime::now())
    .settings(settings)
    .build()?
    .into();

// Convert the HTTP request into a signable request.
let signable = SignableRequest::new(
    "GET",
    "https://some-endpoint.some-region.amazonaws.com",
    std::iter::empty(),
    SignableBody::UnsignedPayload
)?;

// Sign and then apply the signature to the request.
let (signing_instructions, _signature) = sign(signable, &params)?.into_parts();

let mut my_req = http::Request::new("...");
signing_instructions.apply_to_request_http1x(&mut my_req);

```

## 유닛 테스트

AWS SDK for Rust 프로젝트에서 단위 테스트를 구현할 수 있는 방법은 여러 가지가 있지만 몇 가지 권장 사항은 다음과 같습니다.

- [mockall](#) 크레이트 [automock](#) 에서를 사용하여 테스트를 생성하고 실행합니다.
- AWS Smithy 런타임을 사용하여 일반적으로에서 사용되는 표준 HTTP 클라이언트 대신 사용할 수 있는 가짜 HTTP 클라이언트를 [StaticReplayClient](#) 생성합니다 AWS 서비스. 이 클라이언트는 네트워크를 통해 서비스와 통신하는 대신 지정한 HTTP 응답을 반환하므로 테스트는 테스트 목적으로 알려진 데이터를 가져옵니다.

## Mockall을 사용하여 모의 자동 생성

[mockall](#) 크레이트 [automock](#) 에서 널리 사용되는를 사용하여 테스트에 필요한 모의 구현의 대부분을 자동으로 생성할 수 있습니다.

이 예제에서는 라는 사용자 지정 메서드를 테스트합니다 `determine_prefix_file_size()`. 이 메서드는 Amazon S3를 호출하는 사용자 지정 `list_objects()` 래퍼 메서드를 호출합니다. 를 모의하여 Amazon S3 `list_objects()`에 실제로 연락하지 않고도 `determine_prefix_file_size()` 메서드를 테스트할 수 있습니다.

1. 프로젝트 디렉터리에 대한 명령 프롬프트에서 [mockall](#) crate를 종속 항목으로 추가합니다.

```
$ cargo add mockall
```

그러면 `Cargo.toml` 파일의 `[dependencies]` 섹션에 크레이트가 추가됩니다.

2. `mockall` 크레이트의 `automock` 모듈을 포함합니다.

또한 테스트 AWS 서비스 중인 ,이 경우 Amazon S3와 관련된 다른 라이브러리도 포함합니다.

```
use aws_sdk_s3 as s3;
#[allow(unused_imports)]
use mockall::automock;

use s3::operation::list_objects_v2::{ListObjectsV2Error, ListObjectsV2Output};
```

3. 그런 다음, 애플리케이션의 Amazon S3 래퍼 구조 구현 중 사용할 두 가지를 결정하는 코드를 추가합니다.

- 네트워크를 통해 Amazon S3에 액세스하기 위해 작성된 실제입니다.
- 에서 생성된 모의 구현입니다 `mockall`.

이 예제에서는 선택한에 이름이 지정됩니다 `S3`. 선택은 `test` 속성을 기반으로 한 조건부입니다.

```
#[cfg(test)]
pub use MockS3Impl as S3;
#[cfg(not(test))]
pub use S3Impl as S3;
```

4. `S3Impl` 구조는 실제로에 요청을 보내는 Amazon S3 래퍼 구조의 구현입니다 AWS.

- 테스트가 활성화되면 요청이 모의로 전송되고 전송되지 않으므로이 코드는 사용되지 않습니다 AWS. `dead_code` 속성은 `S3Impl` 유형이 사용되지 않는 경우 문제를 보고하지 않도록 리터에 지시합니다.

- 조건은 테스트가 활성화되면 automock 속성을 설정해야 함을 `#[cfg_attr(test, automock)]` 나타냅니다. 이렇게 하면 이름이 인 모의 S3Impl를 생성mockall하도록 지시합니다MockS3Impl.
- 이 예에서 `list_objects()` 메서드는 모의하려는 호출입니다. automock는 자동으로 `expect_list_objects()` 메서드를 생성합니다.

```
#[allow(dead_code)]
pub struct S3Impl {
    inner: s3::Client,
}

#[cfg_attr(test, automock)]
impl S3Impl {
    #[allow(dead_code)]
    pub fn new(inner: s3::Client) -> Self {
        Self { inner }
    }

    #[allow(dead_code)]
    pub async fn list_objects(
        &self,
        bucket: &str,
        prefix: &str,
        continuation_token: Option<String>,
    ) -> Result<ListObjectsV2Output, s3::error::SdkError<ListObjectsV2Error>> {
        self.inner
            .list_objects_v2()
            .bucket(bucket)
            .prefix(prefix)
            .set_continuation_token(continuation_token)
            .send()
            .await
    }
}
```

## 5. 라는 모듈에서 테스트 함수를 생성합니다test.

- 조건은 test 속성이 인 경우가 테스트 모듈을 빌드mockall해야 함을 `#[cfg(test)]` 나타냅니다true.

```

#[cfg(test)]
mod test {
    use super::*;
    use mockall::predicate::eq;

    #[tokio::test]
    async fn test_single_page() {
        let mut mock = MockS3Impl::default();
        mock.expect_list_objects()
            .with(eq("test-bucket"), eq("test-prefix"), eq(None))
            .return_once(|_, _, _| {
                Ok(ListObjectsV2Output::builder()
                    .set_contents(Some(vec![
                        // Mock content for ListObjectsV2 response
                        s3::types::Object::builder().size(5).build(),
                        s3::types::Object::builder().size(2).build(),
                    ]))
                    .build())
            });

        // Run the code we want to test with it
        let size = determine_prefix_file_size(mock, "test-bucket", "test-prefix")
            .await
            .unwrap();

        // Verify we got the correct total size back
        assert_eq!(7, size);
    }

    #[tokio::test]
    async fn test_multiple_pages() {
        // Create the Mock instance with two pages of objects now
        let mut mock = MockS3Impl::default();
        mock.expect_list_objects()
            .with(eq("test-bucket"), eq("test-prefix"), eq(None))
            .return_once(|_, _, _| {
                Ok(ListObjectsV2Output::builder()
                    .set_contents(Some(vec![
                        // Mock content for ListObjectsV2 response
                        s3::types::Object::builder().size(5).build(),
                        s3::types::Object::builder().size(2).build(),
                    ]))
                    .build())
            });
    }
}

```

```

        .set_next_continuation_token(Some("next".to_string()))
        .build()
    });
mock.expect_list_objects()
    .with(
        eq("test-bucket"),
        eq("test-prefix"),
        eq(Some("next".to_string()))
    )
    .return_once(|_, _, _| {
        Ok(ListObjectsV2Output::builder()
            .set_contents(Some(vec![
                // Mock content for ListObjectsV2 response
                s3::types::Object::builder().size(3).build(),
                s3::types::Object::builder().size(9).build(),
            ]))
            .build())
    });

// Run the code we want to test with it
let size = determine_prefix_file_size(mock, "test-bucket", "test-prefix")
    .await
    .unwrap();

assert_eq!(19, size);
}
}

```

- 각 테스트는 `let mut mock = MockS3Impl::default();`를 사용하여의 mock 인스턴스를 생성합니다MockS3Impl.
  - 모의 `expect_list_objects()` 메서드(에 의해 자동으로 생성됨automock)를 사용하여 `list_objects()` 메서드가 코드의 다른 곳에서 사용되는 경우에 대한 예상 결과를 설정합니다.
  - 기대치가 설정되면 이를 사용하여를 호출하여 함수를 테스트합니다determine\_prefix\_file\_size(). 반환된 값은 어설션을 사용하여 올바른지 확인합니다.
6. `determine_prefix_file_size()` 함수는 Amazon S3 래퍼를 사용하여 접두사 파일의 크기를 가져옵니다.

```

#[allow(dead_code)]
pub async fn determine_prefix_file_size(

```

```

// Now we take a reference to our trait object instead of the S3 client
// s3_list: ListObjectsService,
s3_list: S3,
bucket: &str,
prefix: &str,
) -> Result<usize, s3::Error> {
    let mut next_token: Option<String> = None;
    let mut total_size_bytes = 0;
    loop {
        let result = s3_list
            .list_objects(bucket, prefix, next_token.take())
            .await?;

        // Add up the file sizes we got back
        for object in result.contents() {
            total_size_bytes += object.size().unwrap_or(0) as usize;
        }

        // Handle pagination, and break the loop if there are no more pages
        next_token = result.next_continuation_token.clone();
        if next_token.is_none() {
            break;
        }
    }
    Ok(total_size_bytes)
}

```

유형은 HTTP 요청을 할 `MockS3Impl` 때 `S3Impl` 및를 모두 지원하기 위해 래핑된 SDK for Rust 함수를 호출하는 데 `S3` 사용됩니다. 에서 자동으로 생성된 모의는 테스트가 활성화될 때 테스트 실패를 `mockall` 보고합니다.

GitHub에서 [이러한 예제의 전체 코드를 볼](#) 수 있습니다.

## 정적 재생을 사용하여 HTTP 트래픽 시뮬레이션

`aws-smithy-runtime` 크레이트에는 라는 테스트 유틸리티 클래스가 포함되어 있습니다 `StaticReplayClient`. AWS 서비스 객체를 생성할 때 기본 HTTP 클라이언트 대신이 HTTP 클라이언트 클래스를 지정할 수 있습니다.

를 초기화 `StaticReplayClient` 할 때 HTTP 요청 및 응답 페어 목록을 `ReplayEvent` 객체로 제공합니다. 테스트가 실행되는 동안 각 HTTP 요청이 기록되고 클라이언트는 `ReplayEvent` 이벤트 목록

의 다음에 있는 다음 HTTP 응답을 HTTP 클라이언트의 응답으로 반환합니다. 이렇게 하면 네트워크 연결 없이 알려진 데이터를 사용하여 테스트를 실행할 수 있습니다.

## 정적 재생 사용

정적 재생을 사용하려면 래퍼를 사용할 필요가 없습니다. 대신, 테스트에서 사용할 데이터에 대해 실제 네트워크 트래픽이 어떤 모습이어야 하는지 결정하고 SDK가 AWS 서비스 클라이언트로부터 요청을 발행할 때마다 사용할 `StaticReplayClient` 트래픽 데이터에 제공합니다.

### Note

AWS CLI 및 많은 네트워크 트래픽 분석기와 패킷 스니퍼 도구를 포함하여 예상 네트워크 트래픽을 수집하는 방법에는 여러 가지가 있습니다.

- 예상 HTTP 요청과 반환해야 하는 응답을 지정하는 `ReplayEvent` 객체 목록을 생성합니다.
- 이전 단계에서 생성한 HTTP 트랜잭션 목록을 `StaticReplayClient` 사용하여 생성합니다.
- 를 객체의 `StaticReplayClient`로 지정하여 AWS 클라이언트에 대한 구성 `Config` 객체를 생성합니다 `http_client`.
- 이전 단계에서 생성한 구성을 사용하여 AWS 서비스 클라이언트 객체를 생성합니다.
- 를 사용하도록 구성된 서비스 객체를 사용하여 테스트하려는 작업을 수행합니다 `StaticReplayClient`. SDK가 API 요청을 보낼 때마다 목록의 AWS 다음 응답이 사용됩니다.

### Note

전송된 요청이 `ReplayEvent` 객체 벡터의 응답과 일치하지 않더라도 목록의 다음 응답은 항상 반환됩니다.

- 원하는 모든 요청이 이루어지면 `StaticReplayClient.assert_requests_match()` 함수를 호출하여 SDK에서 보낸 요청이 `ReplayEvent` 객체 목록의 요청과 일치하는지 확인합니다.

## 예제

이전 예제에서 동일한 `determine_prefix_file_size()` 함수에 대한 테스트를 살펴보되, 모의 대신 정적 재생을 사용하겠습니다.

1. 프로젝트 디렉터리에 대한 명령 프롬프트에서 [aws-smithy-runtime](#) crate를 종속 항목으로 추가합니다.



```
$ cargo add aws-smithy-runtime --features test-util
```

그러면 Cargo.toml 파일의 [dependencies] 섹션에 크레딧이 추가됩니다.

2. 소스 파일에 필요한 aws\_smithy\_runtime 유형을 포함합니다.

```
use aws_smithy_runtime::client::http::test_util::{ReplayEvent, StaticReplayClient};
use aws_smithy_types::body::SdkBody;
```

3. 테스트는 테스트 중에 발생해야 하는 각 HTTP 트랜잭션을 나타내는 ReplayEvent 구조를 생성하는 것으로 시작됩니다. 각 이벤트에는 HTTP 요청 객체와 일반적으로 회신할 정보를 나타내는 HTTP 응답 객체 AWS 서비스 가 포함됩니다. 이러한 이벤트는에 대한 호출로 전달됩니다StaticReplayClient::new().

```
let page_1 = ReplayEvent::new(
    http::Request::builder()
        .method("GET")
        .uri("https://test-bucket.s3.us-east-1.amazonaws.com/?list-
type=2&prefix=test-prefix")
        .body(SdkBody::empty())
        .unwrap(),
    http::Response::builder()
        .status(200)
        .body(SdkBody::from(include_str!("./testing/
response_multi_1.xml")))
        .unwrap(),
);
let page_2 = ReplayEvent::new(
    http::Request::builder()
        .method("GET")
        .uri("https://test-bucket.s3.us-east-1.amazonaws.com/?list-
type=2&prefix=test-prefix&continuation-token=next")
        .body(SdkBody::empty())
        .unwrap(),
    http::Response::builder()
        .status(200)
        .body(SdkBody::from(include_str!("./testing/
response_multi_2.xml")))
        .unwrap(),
);
let replay_client = StaticReplayClient::new(vec![page_1, page_2]);
```

결과에는 저장됩니다 `replay_client`. 이는 SDK for Rust에서 클라이언트 구성에 지정하여 사용할 수 있는 HTTP 클라이언트를 나타냅니다.

4. Amazon S3 클라이언트를 생성하려면 클라이언트 클래스의 `from_conf()` 함수를 호출하여 구성 객체를 사용하여 클라이언트를 생성합니다.

```
let client: s3::Client = s3::Client::from_conf(
    s3::Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(make_s3_test_credentials())
        .region(s3::config::Region::new("us-east-1"))
        .http_client(replay_client.clone())
        .build(),
);
```

구성 객체는 빌더의 `http_client()` 메서드를 사용하여 지정되고 자격 증명은 `credentials_provider()` 메서드를 사용하여 지정됩니다. 자격 증명은 라는 함수를 사용하여 생성되며 `make_s3_test_credentials()`, 이 함수는 가짜 자격 증명 구조를 반환합니다.

```
fn make_s3_test_credentials() -> s3::config::Credentials {
    s3::config::Credentials::new(
        "ATESTCLIENT",
        "astestsecretkey",
        Some("atestsessiontoken".to_string()),
        None,
        "",
    )
}
```

이러한 자격 증명은 실제로 전송되지 않으므로 유효하지 않아도 됩니다 AWS.

5. 테스트가 필요한 함수를 호출하여 테스트를 실행합니다. 이 예제에서 해당 함수의 이름은 `determine_prefix_file_size()`. 첫 번째 파라미터는 요청에 사용할 Amazon S3 클라이언트 객체입니다. 따라서를 사용하여 생성된 클라이언트를 지정 `StaticReplayClient` 하여 네트워크를 통해 나가지 않고에서 요청을 처리합니다.

```
let size = determine_prefix_file_size(client, "test-bucket", "test-prefix")
    .await
    .unwrap();

assert_eq!(19, size);
```

```
replay_client.assert_requests_match(&[]);
```

에 대한 호출 `determine_prefix_file_size()`이 완료되면 어설트를 사용하여 반환된 값이 예상 값과 일치하는지 확인합니다. 그런 다음 `StaticReplayClient` 메서드 `assert_requests_match()` 함수를 호출합니다. 이 함수는 기록된 HTTP 요청을 스캔하고 재생 클라이언트를 생성할 때 제공된 `ReplayEvent` 객체 배열에 지정된 것과 모두 일치하는지 확인합니다.

GitHub에서 [이러한 예제의 전체 코드를 볼](#) 수 있습니다.

## Waiters

웨이터는 원하는 상태에 도달할 때까지 또는 리소스가 원하는 상태로 전환되지 않을 것으로 확인될 때까지 리소스를 폴링하는 데 사용되는 클라이언트 측 추상화입니다. 이는 Amazon Simple Storage Service와 같이 최종적으로 일관된 서비스 또는 Amazon Elastic Compute Cloud와 같이 비동기적으로 리소스를 생성하는 서비스를 사용할 때 일반적인 작업입니다. 리소스의 상태를 지속적으로 폴링하는 로직을 작성하는 것은 번거롭고 오류가 발생하기 쉽습니다. 웨이터의 목표는 이 책임을 고객 코드에서 옮기는 것입니다. AWS SDK for Rust이 코드는 AWS 작업의 타이밍 측면에 대한 심층적인 지식을 갖추고 있습니다.

AWS 서비스는 `<service>::waiters` 모듈을 포함하여 웨이터를 지원합니다.

- `<service>::client::Waiters` 특성은 클라이언트에 대한 웨이터 메서드를 제공합니다. 메서드는 `Client` 구조체에 구현됩니다. 모든 웨이터 메서드는의 표준 명명 규칙을 따릅니다. `wait_until_<Condition>`
- Amazon S3의 경우이 특성은 [입니다 `aws\_sdk\_s3::client::Waiters`](#).

다음 예제에서는 Amazon S3를 사용합니다. 그러나 개념은 하나 이상의 웨이터 AWS 서비스 가 정의된에 대해 동일합니다.

다음 코드 예제에서는 폴링 로직을 작성하는 대신 웨이터 함수를 사용하여 버킷이 생성된 후 존재할 때까지 기다리는 방법을 보여줍니다.

```
use std::time::Duration;
use aws_config::BehaviorVersion;
// Import Waiters trait to get `wait_until_<Condition>` methods on Client.
use aws_sdk_s3::client::Waiters;
```

```
let config = aws_config::defaults(BehaviorVersion::latest())
    .load()
    .await;

let s3 = aws_sdk_s3::Client::new(&config);

// This initiates creating an S3 bucket and potentially returns before the bucket
// exists.
s3.create_bucket()
    .bucket("my-bucket")
    .send()
    .await?;

// When this function returns, the bucket either exists or an error is propagated.
s3.wait_until_bucket_exists()
    .bucket("my-bucket")
    .wait(Duration::from_secs(5))
    .await?;

// The bucket now exists.
```

### Note

각 대기 메서드는 원하는 조건 또는 오류에 도달한 후 최종 응답에서 가져오는 데 사용할 수 `Result<FinalPoll<...>, WaiterError<...>>` 있는 반환합니다. 자세한 내용은 Rust API 설명서의 [FinalPoll](#) 및 [WaiterError](#)를 참조하세요.

# SDK for Rust 코드 예제

이 주제의 코드 예제에서는 AWS SDK for Rust를 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 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 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 Rust를 사용한 API Gateway 예제](#)
- [SDK for Rust를 사용한 API Gateway Management API 예제](#)
- [SDK for Rust를 사용한 Application Auto Scaling 예제](#)
- [SDK for Rust를 사용한 Aurora 예제](#)
- [SDK for Rust를 사용한 Auto Scaling 예제](#)
- [SDK for Rust를 사용한 Amazon Bedrock 런타임 예제](#)
- [SDK for Rust를 사용한 Amazon Cognito 자격 증명 공급자 예제](#)
- [SDK for Rust를 사용한 Amazon Cognito Sync 예제](#)
- [SDK for Rust를 사용한 Firehose 예제](#)
- [SDK for Rust를 사용한 Amazon DocumentDB 예제](#)
- [SDK for Rust를 사용한 DynamoDB 예제](#)
- [SDK for Rust를 사용한 Amazon EBS 예제](#)
- [SDK for Rust를 사용한 Amazon EC2 예제](#)
- [SDK for Rust를 사용한 Amazon ECR 예제](#)
- [SDK for Rust를 사용한 Amazon ECS 예제](#)
- [SDK for Rust를 사용한 Amazon EKS 예제](#)

- [AWS Glue SDK for Rust를 사용한 예제](#)
- [SDK for Rust를 사용한 IAM 예제](#)
- [AWS IoT SDK for Rust를 사용한 예제](#)
- [SDK for Rust를 사용한 Kinesis 예제](#)
- [AWS KMS SDK for Rust를 사용한 예제](#)
- [SDK for Rust를 사용한 Lambda 예제](#)
- [SDK for Rust를 사용한 MediaLive 예제](#)
- [SDK for Rust를 사용한 MediaPackage 예제](#)
- [SDK for Rust를 사용한 Amazon MSK 예제](#)
- [SDK for Rust를 사용한 Amazon Polly 예제](#)
- [SDK for Rust를 사용한 QLDB 예제](#)
- [SDK for Rust를 사용한 Amazon RDS 예제](#)
- [SDK for Rust를 사용한 Amazon RDS Data Service 예제](#)
- [SDK for Rust를 사용한 Amazon Rekognition 예제](#)
- [SDK for Rust를 사용한 Route 53 예제](#)
- [SDK for Rust를 사용한 Amazon S3 예제](#)
- [SDK for Rust를 사용한 SageMaker AI 예제](#)
- [SDK for Rust를 사용한 Secrets Manager](#)
- [SDK for Rust를 사용한 Amazon SES API v2 예제](#)
- [SDK for Rust를 사용한 Amazon SNS 예제](#)
- [SDK for Rust를 사용한 Amazon SQS 예제](#)
- [AWS STS SDK for Rust를 사용한 예제](#)
- [SDK for Rust를 사용한 Systems Manager 예제](#)
- [SDK for Rust를 사용한 Amazon Transcribe 예제](#)

## SDK for Rust를 사용한 API Gateway 예제

다음 코드 예제에서는 API Gateway와 함께 AWS SDK for Rust를 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.

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

AWS 커뮤니티 기여는 여러 팀이 생성하고 유지 관리하는 예입니다 AWS. 피드백을 제공하려면 연결된 리포지토리에 제공된 메커니즘을 사용합니다.

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

주제

- [작업](#)
- [시나리오](#)
- [AWS 커뮤니티 기여](#)

## 작업

### GetRestApis

다음 코드 예시에서는 GetRestApis을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

리전의 Amazon API Gateway REST API를 표시합니다.

```
async fn show_apis(client: &Client) -> Result<(), Error> {
    let resp = client.get_rest_apis().send().await?;

    for api in resp.items() {
        println!("ID:          {}", api.id().unwrap_or_default());
    }
}
```

```
println!("Name:      {}", api.name().unwrap_or_default());
println!("Description: {}", api.description().unwrap_or_default());
println!("Version:     {}", api.version().unwrap_or_default());
println!(
    "Created:      {}",
    api.created_date().unwrap().to_chrono_utc()?
);
println!();
}

Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetRestApis](#)를 참조하십시오.

## 시나리오

### 사진을 관리하기 위한 서버리스 애플리케이션 만들기

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

#### SDK for Rust

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 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS



## AWS 커뮤니티 기여

### 서버리스 애플리케이션 빌드 및 테스트

다음 코드 예제는 Lambda 및 DynamoDB와 함께 API Gateway를 사용하여 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

#### SDK for Rust

Rust SDK를 사용하여 Lambda 및 DynamoDB가 포함된 API 게이트웨이로 구성된 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

## SDK for Rust를 사용한 API Gateway Management API 예제

다음 코드 예제에서는 API Gateway Management API와 함께 AWS SDK for Rust를 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

#### PostToConnection

다음 코드 예시에서는 PostToConnection을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```

async fn send_data(
    client: &aws_sdk_apigatewaymanagement::Client,
    con_id: &str,
    data: &str,
) -> Result<(), aws_sdk_apigatewaymanagement::Error> {
    client
        .post_to_connection()
        .connection_id(con_id)
        .data(Blob::new(data))
        .send()
        .await?;

    Ok(())
}

let endpoint_url = format!(
    "https://{api_id}.execute-api.{region}.amazonaws.com/{stage}",
    api_id = api_id,
    region = region,
    stage = stage
);

let shared_config = aws_config::from_env().region(region_provider).load().await;
let api_management_config = config::Builder::from(&shared_config)
    .endpoint_url(endpoint_url)
    .build();
let client = Client::from_conf(api_management_config);

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [PostToConnection](#)을 참조하십시오.

## SDK for Rust를 사용한 Application Auto Scaling 예제

다음 코드 예제에서는 Application Auto Scaling과 함께 AWS SDK for Rust를 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

#### DescribeScalingPolicies

다음 코드 예시에서는 DescribeScalingPolicies을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn show_policies(client: &Client) -> Result<(), Error> {
    let response = client
        .describe_scaling_policies()
        .service_namespace(ServiceNamespace::Ec2)
        .send()
        .await?;
    println!("Auto Scaling Policies:");
    for policy in response.scaling_policies() {
        println!("{:?}\n", policy);
    }
}
```

```
println!("Next token: {:?}", response.next_token());

Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeScalingPolicies](#)를 참조하십시오.

## SDK for Rust를 사용한 Aurora 예제

다음 코드 예제에서는 AWS SDK for Rust를 Aurora와 함께 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.

시작

Hello Aurora

다음 코드 예제에서는 Aurora를 사용하여 시작하는 방법을 보여줍니다.

SDK for Rust

### Note

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

```
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
```

```

impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}

impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
        println!("\t Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeDBClusters](#)를 참조하십시오.

## 주제

- [기본 사항](#)
- [작업](#)

## 기본 사항

### 기본 사항 알아보기

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 사용자 지정 Aurora DB 클러스터 파라미터 그룹을 만들고 파라미터 값을 설정합니다.
- 파라미터 그룹을 사용하는 DB 클러스터를 생성합니다.
- 데이터베이스가 포함된 DB 인스턴스를 생성합니다.
- DB 클러스터의 스냅샷을 만든 다음, 리소스를 정리합니다.

### SDK for Rust

#### Note

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

Aurora 시나리오의 시나리오별 함수가 들어 있는 라이브러리입니다.

```
use phf::{phf_set, Set};
use secrecy::SecretString;
use std::{collections::HashMap, fmt::Display, time::Duration};

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

    operation::create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
    types::{DbCluster, DbClusterParameterGroup, DbClusterSnapshot, DbInstance,
    Parameter},
};
use sdk_examples_test_utils::waiter::Waiter;
use tracing::{info, trace, warn};

const DB_ENGINE: &str = "aurora-mysql";
const DB_CLUSTER_PARAMETER_GROUP_NAME: &str = "RustSDKCodeExamplesDBParameterGroup";
const DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION: &str =
    "Parameter Group created by Rust SDK Code Example";
```

```

const DB_CLUSTER_IDENTIFIER: &str = "RustSDKCodeExamplesDBCluster";
const DB_INSTANCE_IDENTIFIER: &str = "RustSDKCodeExamplesDBInstance";

static FILTER_PARAMETER_NAMES: Set<&'static str> = phf_set! {
    "auto_increment_offset",
    "auto_increment_increment",
};

#[derive(Debug, PartialEq, Eq)]
struct MetadataError {
    message: Option<String>,
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(String::from),
            code: err.code().map(String::from),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("{}", code),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{} ({})", message, code),
        };
        write!(f, "{}", display)
    }
}

#[derive(Debug, PartialEq, Eq)]
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {

```

```

        message: message.into(),
        context: None,
    }
}

pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) -> Self {
    ScenarioError {
        message: message.into(),
        context: Some(MetadataError::from(err)),
    }
}

impl std::error::Error for ScenarioError {}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

// Parse the ParameterName, Description, and AllowedValues values and display them.
#[derive(Debug)]
pub struct AuroraScenarioParameter {
    name: String,
    allowed_values: String,
    current_value: String,
}

impl Display for AuroraScenarioParameter {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "{}: {} (allowed: {})",
            self.name, self.current_value, self.allowed_values
        )
    }
}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {

```



```

        name: value.parameter_name.unwrap_or_default(),
        allowed_values: value.allowed_values.unwrap_or_default(),
        current_value: value.parameter_value.unwrap_or_default(),
    }
}

pub struct AuroraScenario {
    rds: crate::rds::Rds,
    engine_family: Option<String>,
    engine_version: Option<String>,
    instance_class: Option<String>,
    db_cluster_parameter_group: Option<DbClusterParameterGroup>,
    db_cluster_identifier: Option<String>,
    db_instance_identifier: Option<String>,
    username: Option<String>,
    password: Option<SecretString>,
}

impl AuroraScenario {
    pub fn new(client: crate::rds::Rds) -> Self {
        AuroraScenario {
            rds: client,
            engine_family: None,
            engine_version: None,
            instance_class: None,
            db_cluster_parameter_group: None,
            db_cluster_identifier: None,
            db_instance_identifier: None,
            username: None,
            password: None,
        }
    }

    // Get available engine families for Aurora MySQL.
    rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
    'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
        let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
        trace!(versions=?describe_db_engine_versions, "full list of versions");

        if let Err(err) = describe_db_engine_versions {

```

```

        return Err(ScenarioError::new(
            "Failed to retrieve DB Engine Versions",
            &err,
        ));
    };

    let version_count = describe_db_engine_versions
        .as_ref()
        .map(|o| o.db_engine_versions().len())
        .unwrap_or_default();
    info!(version_count, "got list of versions");

    // Create a map of engine families to their available versions.
    let mut versions = HashMap::<String, Vec<String>>::new();
    describe_db_engine_versions
        .unwrap()
        .db_engine_versions()
        .iter()
        .filter_map(
            |v| match (&v.db_parameter_group_family, &v.engine_version) {
                (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                _ => None,
            },
        )
        .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

    Ok(versions)
}

pub async fn get_instance_classes(&self) -> Result<Vec<String>, ScenarioError> {
    let describe_orderable_db_instance_options_items = self
        .rds
        .describe_orderable_db_instance_options(
            DB_ENGINE,
            self.engine_version
                .as_ref()
                .expect("engine version for db instance options")
                .as_str(),
        )
        .await;

    describe_orderable_db_instance_options_items

```

```

        .map(|options| {
            options
                .iter()
                .filter(|o| o.storage_type() == Some("aurora"))
                .map(|o| o.db_instance_class().unwrap_or_default().to_string())
                .collect:::<Vec<String>>())
        })
        .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
    }

    // Select an engine family and create a custom DB cluster parameter group.
    rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
        }
    }
}

```

```
    }
    _ => {
        info!("Created Cluster Parameter Group");
    }
}

Ok(())
}

pub fn set_instance_class(&mut self, instance_class: Option<String>) {
    self.instance_class = instance_class;
}

pub fn set_login(&mut self, username: Option<String>, password:
Option<SecretString>) {
    self.username = username;
    self.password = password;
}

pub async fn connection_string(&self) -> Result<String, ScenarioError> {
    let cluster = self.get_cluster().await?;
    let endpoint = cluster.endpoint().unwrap_or_default();
    let port = cluster.port().unwrap_or_default();
    let username = cluster.master_username().unwrap_or_default();
    Ok(format!("mysql -h {endpoint} -P {port} -u {username} -p"))
}

pub async fn get_cluster(&self) -> Result<DbCluster, ScenarioError> {
    let describe_db_clusters_output = self
        .rds
        .describe_db_clusters(
            self.db_cluster_identifier
                .as_ref()
                .expect("cluster identifier")
                .as_str(),
        )
        .await;
    if let Err(err) = describe_db_clusters_output {
        return Err(ScenarioError::new("Failed to get cluster", &err));
    }

    let db_cluster = describe_db_clusters_output
        .unwrap()
        .db_clusters
```

```

        .and_then(|output| output.first().cloned());

    db_cluster.ok_or_else(|| ScenarioError::with("Did not find the cluster"))
}

// Get the parameter group. rds.DescribeDbClusterParameterGroups
// Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
// Parse the ParameterName, Description, and AllowedValues values and display
them.
pub async fn cluster_parameters(&self) -> Result<Vec<AuroraScenarioParameter>,
ScenarioError> {
    let parameters_output = self
        .rds
        .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
        .await;

    if let Err(err) = parameters_output {
        return Err(ScenarioError::new(
            format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
            &err,
        ));
    }

    let parameters = parameters_output
        .unwrap()
        .into_iter()
        .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
        .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
        .map(AuroraScenarioParameter::from)
        .collect:::<Vec<_>>();

    Ok(parameters)
}

// Modify both the auto_increment_offset and auto_increment_increment parameters
in one call in the custom parameter group. Set their ParameterValue fields to a new
allowable value. rds.ModifyDbClusterParameterGroup.
pub async fn update_auto_increment(
    &self,
    offset: u8,

```

```

        increment: u8,
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    // Get a list of allowed engine versions.
    rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySQL database
and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
    // Get a list of instance classes available for the selected engine and engine
version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql', EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
DBInstanceStatus == 'available'.

```

```
pub async fn start_cluster_and_instance(&mut self) -> Result<(), ScenarioError>
{
    if self.password.is_none() {
        return Err(ScenarioError::with(
            "Must set Secret Password before starting a cluster",
        ));
    }
    let create_db_cluster = self
        .rds
        .create_db_cluster(
            DB_CLUSTER_IDENTIFIER,
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            DB_ENGINE,
            self.engine_version.as_deref().expect("engine version"),
            self.username.as_deref().expect("username"),
            self.password
                .replace(SecretString::new("").to_string())
                .expect("password"),
        )
        .await;
    if let Err(err) = create_db_cluster {
        return Err(ScenarioError::new(
            "Failed to create DB Cluster with cluster group",
            &err,
        ));
    }

    self.db_cluster_identifier = create_db_cluster
        .unwrap()
        .db_cluster
        .and_then(|c| c.db_cluster_identifier);

    if self.db_cluster_identifier.is_none() {
        return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
    }

    info!(
        "Started a db cluster: {}",
        self.db_cluster_identifier
            .as_deref()
            .unwrap_or("Missing ARN")
    );
}
```

```
let create_db_instance = self
    .rds
    .create_db_instance(
        self.db_cluster_identifier.as_deref().expect("cluster name"),
        DB_INSTANCE_IDENTIFIER,
        self.instance_class.as_deref().expect("instance class"),
        DB_ENGINE,
    )
    .await;
if let Err(err) = create_db_instance {
    return Err(ScenarioError::new(
        "Failed to create Instance in DB Cluster",
        &err,
    ));
}

self.db_instance_identifier = create_db_instance
    .unwrap()
    .db_instance
    .and_then(|i| i.db_instance_identifier);

// Cluster creation can take up to 20 minutes to become available
let cluster_max_wait = Duration::from_secs(20 * 60);
let waiter = Waiter::builder().max(cluster_max_wait).build();
while waiter.sleep().await.is_ok() {
    let cluster = self
        .rds
        .describe_db_clusters(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = cluster {
        warn!(?err, "Failed to describe cluster while waiting for ready");
        continue;
    }
}

let instance = self
    .rds
    .describe_db_instance(
        self.db_instance_identifier
            .as_deref()
```



```
        .expect("instance identifier"),
    )
    .await;
if let Err(err) = instance {
    return Err(ScenarioError::new(
        "Failed to find instance for cluster",
        &err,
    ));
}

let instances_available = instance
    .unwrap()
    .db_instances()
    .iter()
    .all(|instance| instance.db_instance_status() == Some("Available"));

let endpoints = self
    .rds
    .describe_db_cluster_endpoints(
        self.db_cluster_identifier
            .as_deref()
            .expect("cluster identifier"),
    )
    .await;

if let Err(err) = endpoints {
    return Err(ScenarioError::new(
        "Failed to find endpoint for cluster",
        &err,
    ));
}

let endpoints_available = endpoints
    .unwrap()
    .db_cluster_endpoints()
    .iter()
    .all(|endpoint| endpoint.status() == Some("available"));

if instances_available && endpoints_available {
    return Ok(());
}

Err(ScenarioError::with("timed out waiting for cluster"))
```

```

}

// Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
// Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until Status
== 'available'.
pub async fn snapshot(&self, name: &str) -> Result<DbClusterSnapshot,
ScenarioError> {
    let id = self.db_cluster_identifier.as_deref().unwrap_or_default();
    let snapshot = self
        .rds
        .snapshot_cluster(id, format!("{id}_{name}").as_str())
        .await;
    match snapshot {
        Ok(output) => match output.db_cluster_snapshot {
            Some(snapshot) => Ok(snapshot),
            None => Err(ScenarioError::with("Missing Snapshot")),
        },
        Err(err) => Err(ScenarioError::new("Failed to create snapshot", &err)),
    }
}

pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
    let mut clean_up_errors: Vec<ScenarioError> = vec![];

    // Delete the instance. rds.DeleteDbInstance.
    let delete_db_instance = self
        .rds
        .delete_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = delete_db_instance {
        let identifier = self
            .db_instance_identifier
            .as_deref()
            .unwrap_or("Missing Instance Identifier");
        let message = format!("failed to delete db instance {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance to delete
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {

```

```

        let describe_db_instances = self.rds.describe_db_instances().await;
        if let Err(err) = describe_db_instances {
            clean_up_errors.push(ScenarioError::new(
                "Failed to check instance state during deletion",
                &err,
            ));
            break;
        }
        let db_instances = describe_db_instances
            .unwrap()
            .db_instances()
            .iter()
            .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
            .cloned()
            .collect:::<Vec<DbInstance>>();

        if db_instances.is_empty() {
            trace!("Delete Instance waited and no instances were found");
            break;
        }
        match db_instances.first().unwrap().db_instance_status() {
            Some("Deleting") => continue,
            Some(status) => {
                info!("Attempting to delete but instances is in {status}");
                continue;
            }
            None => {
                warn!("No status for DB instance");
                break;
            }
        }
    }
}

// Delete the DB cluster. rds.DeleteDbCluster.
let delete_db_cluster = self
    .rds
    .delete_db_cluster(
        self.db_cluster_identifier
            .as_deref()
            .expect("cluster identifier"),
    )
    .await;

```

```

    if let Err(err) = delete_db_cluster {
        let identifier = self
            .db_cluster_identifier
            .as_deref()
            .unwrap_or("Missing DB Cluster Identifier");
        let message = format!("failed to delete db cluster {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance and cluster to fully delete.
        rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_clusters = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
            if let Err(err) = describe_db_clusters {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check cluster state during deletion",
                    &err,
                ));
                break;
            }
            let describe_db_clusters = describe_db_clusters.unwrap();
            let db_clusters = describe_db_clusters.db_clusters();
            if db_clusters.is_empty() {
                trace!("Delete cluster waited and no clusters were found");
                break;
            }
            match db_clusters.first().unwrap().status() {
                Some("Deleting") => continue,
                Some(status) => {
                    info!("Attempting to delete but clusters is in {status}");
                    continue;
                }
                None => {
                    warn!("No status for DB cluster");
                    break;
                }
            }
        }
    }
}

```

```

    }
  }
}

// Delete the DB cluster parameter group. rds.DeleteDbClusterParameterGroup.
let delete_db_cluster_parameter_group = self
  .rds
  .delete_db_cluster_parameter_group(
    self.db_cluster_parameter_group
      .map(|g| {
        g.db_cluster_parameter_group_name
          .unwrap_or_else(||
            DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
          })
      .as_deref()
      .expect("cluster parameter group name"),
  )
  .await;
if let Err(error) = delete_db_cluster_parameter_group {
  clean_up_errors.push(ScenarioError::new(
    "Failed to delete the db cluster parameter group",
    &error,
  ))
}

if clean_up_errors.is_empty() {
  Ok(())
} else {
  Err(clean_up_errors)
}
}
}

#[cfg(test)]
pub mod tests;

```

RDS Client 래퍼 주변의 오토모크를 사용하여 라이브러리를 테스트합니다.

```

use crate::rds::MockRdsImpl;

use super::*;

```

```

use std::io::{Error, ErrorKind};

use assert_matches::assert_matches;
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::DeleteDbClusterOutput,
        delete_db_cluster_parameter_group::DeleteDbClusterParameterGroupOutput,
        delete_db_instance::DeleteDbInstanceOutput,
        describe_db_cluster_endpoints::DescribeDbClusterEndpointsOutput,
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError, DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
DescribeDbInstancesOutput},

describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError, ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
        DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
use secrecy::ExposeSecret;

#[tokio::test]
async fn test_scenario_set_engine() {

```

```

let mut mock_rds = MockRdsImpl::default();

mock_rds
    .expect_create_db_cluster_parameter_group()
    .with(
        eq("RustSDKCodeExamplesDBParameterGroup"),
        eq("Parameter Group created by Rust SDK Code Example"),
        eq("aurora-mysql"),
    )
    .return_once(|_, _, _| {
        Ok(CreateDbClusterParameterGroupOutput::builder()

.db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
        .build())
    });

let mut scenario = AuroraScenario::new(mock_rds);

let set_engine = scenario.set_engine("aurora-mysql", "aurora-mysql8.0").await;

assert_eq!(set_engine, Ok(()));
assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-mysql8.0").await;

    assert!(set_engine.is_err());
}

```

```

}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
            )
        });

```



```

        .engine_version("f1b")
        .build(),
    )
    .db_engine_versions(
        DbEngineVersion::builder()
            .db_parameter_group_family("f2")
            .engine_version("f2a")
            .build(),
    )
    .db_engine_versions(DbEngineVersion::builder().build())
    .build()
});

let scenario = AuroraScenario::new(mock_rds);

let versions_map = scenario.get_engines().await;

assert_eq!(
    versions_map,
    Ok(HashMap::from([
        ("f1".into(), vec!["f1a".into(), "f1b".into()]),
        ("f2".into(), vec!["f2a".into()])
    ]))
);
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

```

```

let versions_map = scenario.get_engines().await;
assert_matches!(
    versions_map,
    Err(ScenarioError { message, context: _ }) if message == "Failed to retrieve
DB Engine Versions"
);
}

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

.db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        })
}

```

```

    });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(
                DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
available instance classes"
    );
}

```

```

    );
}

#[tokio::test]
async fn test_scenario_get_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert!(cluster.is_ok());
}

#[tokio::test]
async fn test_scenario_get_cluster_missing_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()
                .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| Ok(DescribeDbClustersOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

```

```

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if message
== "Did not find the cluster");
}

#[tokio::test]
async fn test_scenario_get_cluster_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

.db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_clusters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if message
== "Failed to get cluster");
}

#[tokio::test]
async fn test_scenario_connection_string() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds

```

```

        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .endpoint("test_endpoint")
                        .port(3306)
                        .master_username("test_username")
                        .build(),
                )
                .build())
        });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
let connection_string = scenario.connection_string().await;

assert_eq!(
    connection_string,
    Ok("mysql -h test_endpoint -P 3306 -u test_username -p".into())
);
}

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")

```

```

        .build(),
    )
    .parameters(Parameter::builder().parameter_name("d").build())
    .build()]);
});

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

let params = scenario.cluster_parameters().await.expect("cluster params");
let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
assert_eq!(
    names,
    vec!["auto_increment_offset", "auto_increment_increment"]
);
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message ==
    "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

```

```

mock_rds
    .expect_modify_db_cluster_parameter_group()
    .withf(|name, params| {
        assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
        assert_eq!(
            params,
            &vec![
                Parameter::builder()
                    .parameter_name("auto_increment_offset")
                    .parameter_value("10")
                    .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                    .build(),
                Parameter::builder()
                    .parameter_name("auto_increment_increment")
                    .parameter_value("20")
                    .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                    .build(),
            ]
        );
        true
    })
    .return_once(|_, _|
Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

let scenario = AuroraScenario::new(mock_rds);

scenario
    .update_auto_increment(10, 20)
    .await
    .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(
                ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
            ))
        })

```



```

        )))
        Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
    ))
});

let scenario = AuroraScenario::new(mock_rds);

let update = scenario.update_auto_increment(10, 20).await;
assert_matches!(update, Err(ScenarioError { message, context: _}) if message ==
"Failed to modify cluster parameter group");
}

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()

```

```

        .db_instance(
            DbInstance::builder()
                .db_cluster_identifier(cluster)
                .db_instance_identifier(name)
                .db_instance_class(class)
                .build(),
        )
        .build())
    });

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_describe_db_instance()
    .with(eq("RustSDKCodeExamplesDBInstance"))
    .return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_cluster_endpoints()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
            .build())
    });

let mut scenario = AuroraScenario::new(mock_rds);

```

```

scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let create = scenario.start_cluster_and_instance().await;
    assert!(create.is_ok());
    assert!(scenario
        .password
        .replace(SecretString::new("BAD SECRET".into()))
        .unwrap()
        .expose_secret()
        .is_empty());
    assert_eq!(
        scenario.db_cluster_identifier,
        Some("RustSDKCodeExamplesDBCluster".into())
    );
});
tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::resume();
let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());

```

```

    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message ==
"Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message ==
"Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
        });
}

```

```

        true
    })
    .return_once(|id, _, _, _, _| {
        Ok(CreateDbClusterOutput::builder()
            .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_create_db_instance()
    .return_once(|_, _, _, _| {
        Err(SdkError::service_error(
            CreateDBInstanceError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "create db instance error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

let create = scenario.start_cluster_and_instance().await;
assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
        });
}

```

```

        true
    })
    .return_once(|id, _, _, _, _, _| {
        Ok(CreateDbClusterOutput::builder()
            .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_create_db_instance()
    .withf(|cluster, name, class, engine| {
        assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
        assert_eq!(name, "RustSDKCodeExamplesDBInstance");
        assert_eq!(class, "m5.large");
        assert_eq!(engine, "aurora-mysql");
        true
    })
    .return_once(|cluster, name, class, _| {
        Ok(CreateDbInstanceOutput::builder()
            .db_instance(
                DbInstance::builder()
                    .db_cluster_identifier(cluster)
                    .db_instance_identifier(name)
                    .db_instance_class(class)
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe cluster error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    })
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)

```

```

        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

mock_rds.expect_describe_db_instance().return_once(|name| {
    Ok(DescribeDbInstancesOutput::builder()
        .db_instances(
            DbInstance::builder()
                .db_instance_identifier(name)
                .db_instance_status("Available")
                .build(),
        )
        .build())
});

mock_rds
    .expect_describe_db_cluster_endpoints()
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let create = scenario.start_cluster_and_instance().await;
    assert!(create.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::resume();
let _ = assertions.await;
}

```

```
#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
            )
        })
    }
```



```

        .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))
    .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
tokio::time::resume();
let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds

```

```
.expect_delete_db_instance()
.with(eq("MockInstance"))
.return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

mock_rds
.expect_describe_db_instances()
.with()
.times(1)
.returning(|| {
    Ok(DescribeDbInstancesOutput::builder()
        .db_instances(
            DbInstance::builder()
                .db_cluster_identifier("MockCluster")
                .db_instance_status("Deleting")
                .build(),
        )
        .build())
})
.with()
.times(1)
.returning(|| {
    Err(SdkError::service_error(
        DescribeDBInstancesError::unhandled(Box::new(Error::new(
            ErrorKind::Other,
            "describe db instances error",
        ))),
        Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
    ))
});

mock_rds
.expect_delete_db_cluster()
.with(eq("MockCluster"))
.return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

mock_rds
.expect_describe_db_clusters()
.with(eq("MockCluster"))
.times(1)
.returning(|id| {
    Ok(DescribeDbClustersOutput::builder()
        .db_clusters(
            DbCluster::builder()
                .db_cluster_identifier(id)

```

```

                .status("Deleting")
                .build(),
            )
            .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe db clusters error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))
    .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_err());
    let errs = clean_up.unwrap_err();
    assert_eq!(errs.len(), 2);
    assert_matches!(errs.first(), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
    assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
});

```

```

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_snapshot() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Ok(CreateDbClusterSnapshotOutput::builder()
                .db_cluster_snapshot(
                    DbClusterSnapshot::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_cluster_snapshot_identifier("MockCluster_MockSnapshot")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert!(create_snapshot.is_ok());
}

#[tokio::test]
async fn test_scenario_snapshot_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))

```

```

        .times(1)
        .return_once(|_, _| {
            Err(SdkError::service_error(
                CreateDBClusterSnapshotError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create snapshot error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
message == "Failed to create snapshot");
}

#[tokio::test]
async fn test_scenario_snapshot_invalid() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| Ok(CreateDbClusterSnapshotOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
message == "Missing Snapshot");
}

```

시나리오를 처음부터 끝까지 실행하는 바이너리입니다. 사용자가 일부 결정을 내릴 수 있도록 인과 이어러를 사용합니다.

```

use std::fmt::Display;

use anyhow::anyhow;

```

```

use aurora_code_examples::{
    aurora_scenario::{AuroraScenario, ScenarioError},
    rds::Rds as RdsClient,
};
use aws_sdk_rds::Client;
use inquire::{validator::StringValidator, CustomUserError};
use secrecy::SecretString;
use tracing::warn;

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    fn new() -> Self {
        Warnings(Vec::with_capacity(5))
    }

    fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:");
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "");
        }
        Ok(())
    }
}

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)

```

```

        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
    }

// Prepare the Aurora Scenario. Prompt for several settings that are optional to the
// Scenario, but that the user should choose for the demo.
// This includes the engine, engine version, and instance class.
async fn prepare_scenario(rds: RdsClient) -> Result<AuroraScenario, anyhow::Error> {
    let mut scenario = AuroraScenario::new(rds);

    // Get available engine families for Aurora MySQL.
    rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
    'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    let available_engines = scenario.get_engines().await;
    if let Err(error) = available_engines {
        return Err(anyhow!("Failed to get available engines: {}", error));
    }
    let available_engines = available_engines.unwrap();

    // Select an engine family and create a custom DB cluster parameter group.
    rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    let engine = select(
        "Select an Aurora engine family",
        available_engines.keys().cloned().collect:::<Vec<String>>(),
        "Invalid engine selection",
    )?;

    let version = select(
        format!("Select an Aurora engine version for {engine}").as_str(),
        available_engines.get(&engine).cloned().unwrap_or_default(),
        "Invalid engine version selection",
    )?;

    let set_engine = scenario.set_engine(engine.as_str(), version.as_str()).await;
    if let Err(error) = set_engine {
        return Err(anyhow!("Could not set engine: {}", error));
    }

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
            )?;

```

```

        "Invalid instance class selection",
    )?;
    scenario.set_instance_class(Some(instance_class))
}
Err(err) => return Err(anyhow!("Failed to get instance classes for engine:
{err}")),
}

Ok(scenario)
}

// Prepare the cluster, creating a custom parameter group overriding some group
parameters based on user input.
async fn prepare_cluster(scenario: &mut AuroraScenario, warnings: &mut Warnings) ->
Result<(), ()> {
    show_parameters(scenario, warnings).await;

    let offset = prompt_number_or_default(warnings, "auto_increment_offset", 5);
    let increment = prompt_number_or_default(warnings, "auto_increment_increment",
3);

    // Modify both the auto_increment_offset and auto_increment_increment parameters
in one call in the custom parameter group. Set their ParameterValue fields to a new
allowable value. rds.ModifyDbClusterParameterGroup.
    let update_auto_increment = scenario.update_auto_increment(offset,
increment).await;

    if let Err(error) = update_auto_increment {
        warnings.push("Failed to update auto increment", error);
        return Err(());
    }

    // Get and display the updated parameters. Specify Source of 'user' to get just
the modified parameters. rds.DescribeDbClusterParameters(Source='user')
    show_parameters(scenario, warnings).await;

    let username = inquire::Text::new("Username for the database (default
'testuser')")
        .with_default("testuser")
        .with_initial_value("testuser")
        .prompt();

    if let Err(error) = username {
        warnings.push(

```



```

        "Failed to get username, using default",
        ScenarioError::with(format!("Error from inquirer: {error}")),
    );
    return Err(());
}
let username = username.unwrap();

let password = inquire::Text::new("Password for the database (minimum 8
characters)")
    .with_validator(|i: &str| {
        if i.len() >= 8 {
            Ok(inquire::validator::Validation::Valid)
        } else {
            Ok(inquire::validator::Validation::Invalid(
                "Password must be at least 8 characters".into(),
            ))
        }
    })
    .prompt();

let password: Option<SecretString> = match password {
    Ok(password) => Some(SecretString::from(password)),
    Err(error) => {
        warnings.push(
            "Failed to get password, using none (and not starting a DB)",
            ScenarioError::with(format!("Error from inquirer: {error}")),
        );
        return Err(());
    }
};

scenario.set_login(Some(username), password);

Ok(())
}

// Start a single instance in the cluster,
async fn run_instance(scenario: &mut AuroraScenario) -> Result<(), ScenarioError> {
    // Create an Aurora DB cluster database cluster that contains a MySQL database
    and uses the parameter group you created.
    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
    DBInstanceStatus == 'available'.
    scenario.start_cluster_and_instance().await?;
}

```

```

    let connection_string = scenario.connection_string().await?;

    println!("Database ready: {connection_string}");

    let _ = inquire::Text::new("Use the database with the connection string. When
you're finished, press enter key to continue.").prompt();

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until Status
    == 'available'.
    let snapshot_name = inquire::Text::new("Provide a name for the snapshot")
        .prompt()
        .unwrap_or(String::from("ScenarioRun"));
    let snapshot = scenario.snapshot(snapshot_name.as_str()).await?;
    println!(
        "Snapshot is available: {}",
        snapshot.db_cluster_snapshot_arn().unwrap_or("Missing ARN")
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
    let rds = RdsClient::new(client);
    let mut scenario = prepare_scenario(rds).await?;

    // At this point, the scenario has things in AWS and needs to get cleaned up.
    let mut warnings = Warnings::new();

    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }
}

// Clean up the instance, cluster, and parameter group, waiting for the instance
and cluster to delete before moving on.
let clean_up = scenario.clean_up().await;

```

```

    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
        }
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
    }
}

#[derive(Clone)]
struct U8Validator {}
impl StringValidator for U8Validator {
    fn validate(&self, input: &str) -> Result<inquire::validator::Validation,
CustomUserError> {
        if input.parse::<u8>().is_err() {
            Ok(inquire::validator::Validation::Invalid(
                "Can't parse input as number".into(),
            ))
        } else {
            Ok(inquire::validator::Validation::Valid)
        }
    }
}

async fn show_parameters(scenario: &AuroraScenario, warnings: &mut Warnings) {
    let parameters = scenario.cluster_parameters().await;

    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
            for parameter in parameters {
                println!("\t{parameter}");
            }
        }
        Err(error) => warnings.push("Could not find cluster parameters", error),
    }
}

```

```

fn prompt_number_or_default(warnings: &mut Warnings, name: &str, default: u8) -> u8
{
    let input = inquire::Text::new(format!("Updated {name}:").as_str())
        .with_validator(U8Validator {})
        .prompt();

    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
            Err(error) => {
                warnings.push(
                    format!("Invalid updated {name} (using {default}
instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}

```

테스트를 위한 오토모킹(automocking)을 허용하는 Amazon RDS 서비스를 둘러싼 래퍼입니다.

```

use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
    },
};

```

```

delete_db_cluster_parameter_group::{
    DeleteDBClusterParameterGroupError, DeleteDbClusterParameterGroupOutput,
},
delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
describe_db_cluster_endpoints::{
    DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
},
describe_db_cluster_parameters::{
    DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
},
describe_db_clusters::{DescribeDBClustersError, DescribeDbClustersOutput},
describe_db_engine_versions::{
    DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
},
describe_db_instances::{DescribeDBInstancesError,
DescribeDbInstancesOutput},

describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
modify_db_cluster_parameter_group::{
    ModifyDBClusterParameterGroupError, ModifyDbClusterParameterGroupOutput,
},
},
types::{OrderableDbInstanceOption, Parameter},
Client as RdsClient,
};
use secrecy::{ExposeSecret, SecretString};

#[cfg(test)]
use mockall::automock;

#[cfg(test)]
pub use MockRdsImpl as Rds;
#[cfg(not(test))]
pub use RdsImpl as Rds;

pub struct RdsImpl {
    pub inner: RdsClient,
}

#[cfg_attr(test, automock)]
impl RdsImpl {
    pub fn new(inner: RdsClient) -> Self {
        RdsImpl { inner }
    }
}

```

```
pub async fn describe_db_engine_versions(
    &self,
    engine: &str,
) -> Result<DescribeDbEngineVersionsOutput,
SdkError<DescribeDBEngineVersionsError>> {
    self.inner
        .describe_db_engine_versions()
        .engine(engine)
        .send()
        .await
}

pub async fn describe_orderable_db_instance_options(
    &self,
    engine: &str,
    engine_version: &str,
) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
{
    self.inner
        .describe_orderable_db_instance_options()
        .engine(engine)
        .engine_version(engine_version)
        .into_paginator()
        .items()
        .send()
        .try_collect()
        .await
}

pub async fn create_db_cluster_parameter_group(
    &self,
    name: &str,
    description: &str,
    family: &str,
) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
{
    self.inner
        .create_db_cluster_parameter_group()
        .db_cluster_parameter_group_name(name)
        .description(description)
        .db_parameter_group_family(family)
```

```
        .send()
        .await
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }
}
```

```
pub async fn create_db_cluster(
    &self,
    name: &str,
    parameter_group: &str,
    engine: &str,
    version: &str,
    username: &str,
    password: SecretString,
) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
    self.inner
        .create_db_cluster()
        .db_cluster_identifier(name)
        .db_cluster_parameter_group_name(parameter_group)
        .engine(engine)
        .engine_version(version)
        .master_username(username)
        .master_user_password(password.expose_secret())
        .send()
        .await
}

pub async fn create_db_instance(
    &self,
    cluster_name: &str,
    instance_name: &str,
    instance_class: &str,
    engine: &str,
) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
    self.inner
        .create_db_instance()
        .db_cluster_identifier(cluster_name)
        .db_instance_identifier(instance_name)
        .db_instance_class(instance_class)
        .engine(engine)
        .send()
        .await
}

pub async fn describe_db_instance(
    &self,
    instance_identifier: &str,
) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
    self.inner
```



```
        .describe_db_instances()
        .db_instance_identifier(instance_identifier)
        .send()
        .await
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

    pub async fn describe_db_cluster_endpoints(
        &self,
        cluster_identifier: &str,
    ) -> Result<DescribeDbClusterEndpointsOutput,
SdkError<DescribeDBClusterEndpointsError>> {
        self.inner
            .describe_db_cluster_endpoints()
            .db_cluster_identifier(cluster_identifier)
            .send()
            .await
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
    }
}
```

```

        .db_instance_identifier(instance_identifier)
        .skip_final_snapshot(true)
        .send()
        .await
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
        SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }
}

```

이 시나리오에 사용된 종속 항목이 있는 Cargo.toml입니다.

```

[package]
name = "aurora-code-examples"
authors = [
    "David Souther <dpsouth@amazon.com>",
]
edition = "2021"
version = "0.1.0"

```

```
# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
```

```
[dependencies]
anyhow = "1.0.75"
assert_matches = "1.5.0"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime-api = { version = "1.0.1" }
aws-sdk-rds = { version = "1.3.0" }
inquire = "0.6.2"
mockall = "0.11.4"
phf = { version = "0.11.2", features = ["std", "macros"] }
sdk-examples-test-utils = { path = "../..../test-utils" }
secrecy = "0.8.0"
tokio = { version = "1.20.1", features = ["full", "test-util"] }
tracing = "0.1.37"
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
```

• API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하십시오.

- [CreateDBCluster](#)
- [CreateDBClusterParameterGroup](#)
- [CreateDBClusterSnapshot](#)
- [CreateDBInstance](#)
- [DeleteDBCluster](#)
- [DeleteDBClusterParameterGroup](#)
- [DeleteDBInstance](#)
- [DescribeDBClusterParameterGroups](#)
- [DescribeDBClusterParameters](#)
- [DescribeDBClusterSnapshots](#)
- [DescribeDBClusters](#)
- [DescribeDBEngineVersions](#)
- [DescribeDBInstances](#)
- [DescribeOrderableDBInstanceOptions](#)
- [ModifyDBClusterParameterGroup](#)

## 작업

### CreateDBCluster

다음 코드 예시에서는 CreateDBCluster을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
// Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
// Create an Aurora DB cluster database cluster that contains a MySQL database
and uses the parameter group you created.
// Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
// Get a list of instance classes available for the selected engine and engine
version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql', EngineVersion=).

// Create a database instance in the cluster.
// Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
DBInstanceStatus == 'available'.
pub async fn start_cluster_and_instance(&mut self) -> Result<(), ScenarioError>
{
    if self.password.is_none() {
        return Err(ScenarioError::with(
            "Must set Secret Password before starting a cluster",
        ));
    }
    let create_db_cluster = self
        .rds
        .create_db_cluster(
            DB_CLUSTER_IDENTIFIER,
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            DB_ENGINE,
            self.engine_version.as_deref().expect("engine version"),
            self.username.as_deref().expect("username"),
```

```
        self.password
            .replace(SecretString::new("").to_string()))
            .expect("password"),
    )
    .await;
if let Err(err) = create_db_cluster {
    return Err(ScenarioError::new(
        "Failed to create DB Cluster with cluster group",
        &err,
    ));
}

self.db_cluster_identifier = create_db_cluster
    .unwrap()
    .db_cluster
    .and_then(|c| c.db_cluster_identifier);

if self.db_cluster_identifier.is_none() {
    return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
}

info!(
    "Started a db cluster: {}",
    self.db_cluster_identifier
        .as_deref()
        .unwrap_or("Missing ARN")
);

let create_db_instance = self
    .rds
    .create_db_instance(
        self.db_cluster_identifier.as_deref().expect("cluster name"),
        DB_INSTANCE_IDENTIFIER,
        self.instance_class.as_deref().expect("instance class"),
        DB_ENGINE,
    )
    .await;
if let Err(err) = create_db_instance {
    return Err(ScenarioError::new(
        "Failed to create Instance in DB Cluster",
        &err,
    ));
}
```

```
self.db_instance_identifier = create_db_instance
    .unwrap()
    .db_instance
    .and_then(|i| i.db_instance_identifier);

// Cluster creation can take up to 20 minutes to become available
let cluster_max_wait = Duration::from_secs(20 * 60);
let waiter = Waiter::builder().max(cluster_max_wait).build();
while waiter.sleep().await.is_ok() {
    let cluster = self
        .rds
        .describe_db_clusters(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = cluster {
        warn!(?err, "Failed to describe cluster while waiting for ready");
        continue;
    }

    let instance = self
        .rds
        .describe_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = instance {
        return Err(ScenarioError::new(
            "Failed to find instance for cluster",
            &err,
        ));
    }

    let instances_available = instance
        .unwrap()
        .db_instances()
        .iter()
        .all(|instance| instance.db_instance_status() == Some("Available"));
}
```

```
    let endpoints = self
        .rds
        .describe_db_cluster_endpoints(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = endpoints {
        return Err(ScenarioError::new(
            "Failed to find endpoint for cluster",
            &err,
        ));
    }

    let endpoints_available = endpoints
        .unwrap()
        .db_cluster_endpoints()
        .iter()
        .all(|endpoint| endpoint.status() == Some("available"));

    if instances_available && endpoints_available {
        return Ok(());
    }
}

Err(ScenarioError::with("timed out waiting for cluster"))
}

pub async fn create_db_cluster(
    &self,
    name: &str,
    parameter_group: &str,
    engine: &str,
    version: &str,
    username: &str,
    password: SecretString,
) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
    self.inner
        .create_db_cluster()
        .db_cluster_identifier(name)
        .db_cluster_parameter_group_name(parameter_group)
```

```

        .engine(engine)
        .engine_version(version)
        .master_username(username)
        .master_user_password(password.expose_secret())
        .send()
        .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                )
            )
        });
}

```



```

        .db_instance_class(class)
        .build(),
    )
    .build())
});

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_describe_db_instance()
    .with(eq("RustSDKCodeExamplesDBInstance"))
    .return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_cluster_endpoints()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
        .build())
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

```

```

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let create = scenario.start_cluster_and_instance().await;
    assert!(create.is_ok());
    assert!(scenario
        .password
        .replace(SecretString::new("BAD SECRET".into()))
        .unwrap()
        .expose_secret()
        .is_empty());
    assert_eq!(
        scenario.db_cluster_identifier,
        Some("RustSDKCodeExamplesDBCluster".into())
    );
});
tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::resume();
let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;

```

```

    assert_matches!(create, Err(ScenarioError { message, context: _}) if message ==
"Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

```

```

        .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
        .build()
    });

mock_rds
    .expect_create_db_instance()
    .return_once(|_, _, _, _| {
        Err(SdkError::service_error(
            CreateDBInstanceError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "create db instance error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

let create = scenario.start_cluster_and_instance().await;
assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

```

```

        .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
        .build()
    });

mock_rds
    .expect_create_db_instance()
    .withf(|cluster, name, class, engine| {
        assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
        assert_eq!(name, "RustSDKCodeExamplesDBInstance");
        assert_eq!(class, "m5.large");
        assert_eq!(engine, "aurora-mysql");
        true
    })
    .return_once(|cluster, name, class, _| {
        Ok(CreateDbInstanceOutput::builder()
            .db_instance(
                DbInstance::builder()
                    .db_cluster_identifier(cluster)
                    .db_instance_identifier(name)
                    .db_instance_class(class)
                    .build(),
            )
            .build()
        ));

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe cluster error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    })
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
            .build()
        ));
    });

```

```

    });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()
                .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateDBCluster](#)를 참조하십시오.

## CreateDBClusterParameterGroup

다음 코드 예시에서는 CreateDBClusterParameterGroup을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
// Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
    self.engine_family = Some(engine.to_string());
    self.engine_version = Some(version.to_string());
    let create_db_cluster_parameter_group = self
        .rds
        .create_db_cluster_parameter_group(
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
            engine,
        )
        .await;

    match create_db_cluster_parameter_group {
        Ok(CreateDbClusterParameterGroupOutput {
            db_cluster_parameter_group: None,
            ..
        }) => {
            return Err(ScenarioError::with(
                "CreateDBClusterParameterGroup had empty response",
            ));
        }
        Err(error) => {
            if error.code() == Some("DBParameterGroupAlreadyExists") {
                info!("Cluster Parameter Group already exists, nothing to do");
            } else {
                return Err(ScenarioError::new(
                    "Could not create Cluster Parameter Group",
                    &error,
                ));
            }
        }
    }
}
```

```

        ));
    }
}
_ => {
    info!("Created Cluster Parameter Group");
}
}

Ok(())
}

pub async fn create_db_cluster_parameter_group(
    &self,
    name: &str,
    description: &str,
    family: &str,
) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
{
    self.inner
        .create_db_cluster_parameter_group()
        .db_cluster_parameter_group_name(name)
        .description(description)
        .db_parameter_group_family(family)
        .send()
        .await
}

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

                .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        })
}

```



```

    });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                )
            )
        })
}

```

```

        ),
        Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
    ))
});

let mut scenario = AuroraScenario::new(mock_rds);

let set_engine = scenario.set_engine("aurora-mysql", "aurora-mysql8.0").await;

assert!(set_engine.is_err());
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateDBClusterParameterGroup](#)를 참조하십시오.

## CreateDBClusterSnapshot

다음 코드 예시에서는 CreateDBClusterSnapshot을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

// Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
// Create an Aurora DB cluster database cluster that contains a MySQL database
and uses the parameter group you created.
// Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
// Get a list of instance classes available for the selected engine and engine
version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql', EngineVersion=).

// Create a database instance in the cluster.
// Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
DBInstanceStatus == 'available'.

```

```
pub async fn start_cluster_and_instance(&mut self) -> Result<(), ScenarioError>
{
    if self.password.is_none() {
        return Err(ScenarioError::with(
            "Must set Secret Password before starting a cluster",
        ));
    }
    let create_db_cluster = self
        .rds
        .create_db_cluster(
            DB_CLUSTER_IDENTIFIER,
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            DB_ENGINE,
            self.engine_version.as_deref().expect("engine version"),
            self.username.as_deref().expect("username"),
            self.password
                .replace(SecretString::new("").to_string())
                .expect("password"),
        )
        .await;
    if let Err(err) = create_db_cluster {
        return Err(ScenarioError::new(
            "Failed to create DB Cluster with cluster group",
            &err,
        ));
    }

    self.db_cluster_identifier = create_db_cluster
        .unwrap()
        .db_cluster
        .and_then(|c| c.db_cluster_identifier);

    if self.db_cluster_identifier.is_none() {
        return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
    }

    info!(
        "Started a db cluster: {}",
        self.db_cluster_identifier
            .as_deref()
            .unwrap_or("Missing ARN")
    );
}
```

```
let create_db_instance = self
    .rds
    .create_db_instance(
        self.db_cluster_identifier.as_deref().expect("cluster name"),
        DB_INSTANCE_IDENTIFIER,
        self.instance_class.as_deref().expect("instance class"),
        DB_ENGINE,
    )
    .await;
if let Err(err) = create_db_instance {
    return Err(ScenarioError::new(
        "Failed to create Instance in DB Cluster",
        &err,
    ));
}

self.db_instance_identifier = create_db_instance
    .unwrap()
    .db_instance
    .and_then(|i| i.db_instance_identifier);

// Cluster creation can take up to 20 minutes to become available
let cluster_max_wait = Duration::from_secs(20 * 60);
let waiter = Waiter::builder().max(cluster_max_wait).build();
while waiter.sleep().await.is_ok() {
    let cluster = self
        .rds
        .describe_db_clusters(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = cluster {
        warn!(?err, "Failed to describe cluster while waiting for ready");
        continue;
    }

    let instance = self
        .rds
        .describe_db_instance(
            self.db_instance_identifier
                .as_deref()
```

```
                .expect("instance identifier"),
            )
            .await;
        if let Err(err) = instance {
            return Err(ScenarioError::new(
                "Failed to find instance for cluster",
                &err,
            ));
        }

        let instances_available = instance
            .unwrap()
            .db_instances()
            .iter()
            .all(|instance| instance.db_instance_status() == Some("Available"));

        let endpoints = self
            .rds
            .describe_db_cluster_endpoints(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = endpoints {
            return Err(ScenarioError::new(
                "Failed to find endpoint for cluster",
                &err,
            ));
        }

        let endpoints_available = endpoints
            .unwrap()
            .db_cluster_endpoints()
            .iter()
            .all(|endpoint| endpoint.status() == Some("available"));

        if instances_available && endpoints_available {
            return Ok(());
        }
    }

    Err(ScenarioError::with("timed out waiting for cluster"))
}
```

```

    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
        });
}

```

```

        true
    })
    .return_once(|cluster, name, class, _| {
        Ok(CreateDbInstanceOutput::builder()
            .db_instance(
                DbInstance::builder()
                    .db_cluster_identifier(cluster)
                    .db_instance_identifier(name)
                    .db_instance_class(class)
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_describe_db_instance()
    .with(eq("RustSDKCodeExamplesDBInstance"))
    .return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_cluster_endpoints()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()
            .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())

```

```

        .build()
    });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });
}

```



```

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

let create = scenario.start_cluster_and_instance().await;
assert_matches!(create, Err(ScenarioError { message, context: _}) if message ==
"Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build());
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
        });
}

```

```

        assert_eq!(engine, "aurora-mysql");
        assert_eq!(version, "aurora-mysql8.0");
        assert_eq!(username, "test username");
        assert_eq!(password.expose_secret(), "test password");
        true
    })
    .return_once(|id, _, _, _, _| {
        Ok(CreateDbClusterOutput::builder()
            .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_create_db_instance()
    .return_once(|_, _, _, _| {
        Err(SdkError::service_error(
            CreateDBInstanceError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "create db instance error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

let create = scenario.start_cluster_and_instance().await;
assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
        });

```

```

        assert_eq!(engine, "aurora-mysql");
        assert_eq!(version, "aurora-mysql8.0");
        assert_eq!(username, "test username");
        assert_eq!(password.expose_secret(), "test password");
        true
    })
    .return_once(|id, _, _, _, _, _| {
        Ok(CreateDbClusterOutput::builder()
            .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_create_db_instance()
    .withf(|cluster, name, class, engine| {
        assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
        assert_eq!(name, "RustSDKCodeExamplesDBInstance");
        assert_eq!(class, "m5.large");
        assert_eq!(engine, "aurora-mysql");
        true
    })
    .return_once(|cluster, name, class, _| {
        Ok(CreateDbInstanceOutput::builder()
            .db_instance(
                DbInstance::builder()
                    .db_cluster_identifier(cluster)
                    .db_instance_identifier(name)
                    .db_instance_class(class)
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe cluster error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

```

```

        ))
    })
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds.expect_describe_db_instance().return_once(|name| {
    Ok(DescribeDbInstancesOutput::builder()
        .db_instances(
            DbInstance::builder()
                .db_instance_identifier(name)
                .db_instance_status("Available")
                .build(),
        )
        .build())
});

mock_rds
    .expect_describe_db_cluster_endpoints()
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
            .build())
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let create = scenario.start_cluster_and_instance().await;
    assert!(create.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::advance(Duration::from_secs(1)).await;

```

```

    tokio::time::resume();
    let _ = assertions.await;
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateDBClusterSnapshot](#)을 참조하십시오.

## CreateDBInstance

다음 코드 예시에서는 CreateDBInstance을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

// Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
// Create an Aurora DB cluster database cluster that contains a MySQL database
and uses the parameter group you created.
// Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
// Get a list of instance classes available for the selected engine and engine
version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql', EngineVersion=).

// Create a database instance in the cluster.
// Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
DBInstanceStatus == 'available'.
pub async fn start_cluster_and_instance(&mut self) -> Result<(), ScenarioError>
{
    if self.password.is_none() {
        return Err(ScenarioError::with(
            "Must set Secret Password before starting a cluster",
        ));
    }
    let create_db_cluster = self
        .rds
        .create_db_cluster(

```

```

        DB_CLUSTER_IDENTIFIER,
        DB_CLUSTER_PARAMETER_GROUP_NAME,
        DB_ENGINE,
        self.engine_version.as_deref().expect("engine version"),
        self.username.as_deref().expect("username"),
        self.password
            .replace(SecretString::new("").to_string())
            .expect("password"),
    )
    .await;
if let Err(err) = create_db_cluster {
    return Err(ScenarioError::new(
        "Failed to create DB Cluster with cluster group",
        &err,
    ));
}

self.db_cluster_identifier = create_db_cluster
    .unwrap()
    .db_cluster
    .and_then(|c| c.db_cluster_identifier);

if self.db_cluster_identifier.is_none() {
    return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
}

info!(
    "Started a db cluster: {}",
    self.db_cluster_identifier
        .as_deref()
        .unwrap_or("Missing ARN")
);

let create_db_instance = self
    .rds
    .create_db_instance(
        self.db_cluster_identifier.as_deref().expect("cluster name"),
        DB_INSTANCE_IDENTIFIER,
        self.instance_class.as_deref().expect("instance class"),
        DB_ENGINE,
    )
    .await;
if let Err(err) = create_db_instance {

```

```
        return Err(ScenarioError::new(
            "Failed to create Instance in DB Cluster",
            &err,
        ));
    }

    self.db_instance_identifer = create_db_instance
        .unwrap()
        .db_instance
        .and_then(|i| i.db_instance_identifer);

    // Cluster creation can take up to 20 minutes to become available
    let cluster_max_wait = Duration::from_secs(20 * 60);
    let waiter = Waiter::builder().max(cluster_max_wait).build();
    while waiter.sleep().await.is_ok() {
        let cluster = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifer
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = cluster {
            warn!(?err, "Failed to describe cluster while waiting for ready");
            continue;
        }

        let instance = self
            .rds
            .describe_db_instance(
                self.db_instance_identifer
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = instance {
            return Err(ScenarioError::new(
                "Failed to find instance for cluster",
                &err,
            ));
        }
    }
}
```

```

        let instances_available = instance
            .unwrap()
            .db_instances()
            .iter()
            .all(|instance| instance.db_instance_status() == Some("Available"));

        let endpoints = self
            .rds
            .describe_db_cluster_endpoints(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = endpoints {
            return Err(ScenarioError::new(
                "Failed to find endpoint for cluster",
                &err,
            ));
        }

        let endpoints_available = endpoints
            .unwrap()
            .db_cluster_endpoints()
            .iter()
            .all(|endpoint| endpoint.status() == Some("available"));

        if instances_available && endpoints_available {
            return Ok(());
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner

```



```

        .create_db_instance()
        .db_cluster_identifier(cluster_name)
        .db_instance_identifier(instance_name)
        .db_instance_class(instance_class)
        .engine(engine)
        .send()
        .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)

```

```

        .db_instance_identifier(name)
        .db_instance_class(class)
        .build(),
    )
    .build()
});

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_describe_db_instance()
    .with(eq("RustSDKCodeExamplesDBInstance"))
    .return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_cluster_endpoints()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
            .build())
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());

```

```

scenario.password = Some(SecretString::new("test password".into()));

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let create = scenario.start_cluster_and_instance().await;
    assert!(create.is_ok());
    assert!(scenario
        .password
        .replace(SecretString::new("BAD SECRET".into()))
        .unwrap()
        .expose_secret()
        .is_empty());
    assert_eq!(
        scenario.db_cluster_identifier,
        Some("RustSDKCodeExamplesDBCluster".into())
    );
});
tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::resume();
let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
}

```

```

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message ==
"Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {

```

```

        Ok(CreateDbClusterOutput::builder()
            .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_create_db_instance()
    .return_once(|_, _, _, _| {
        Err(SdkError::service_error(
            CreateDBInstanceError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "create db instance error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

let create = scenario.start_cluster_and_instance().await;
assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {

```

```

        Ok(CreateDbClusterOutput::builder()
            .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_create_db_instance()
    .withf(|cluster, name, class, engine| {
        assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
        assert_eq!(name, "RustSDKCodeExamplesDBInstance");
        assert_eq!(class, "m5.large");
        assert_eq!(engine, "aurora-mysql");
        true
    })
    .return_once(|cluster, name, class, _| {
        Ok(CreateDbInstanceOutput::builder()
            .db_instance(
                DbInstance::builder()
                    .db_cluster_identifier(cluster)
                    .db_instance_identifier(name)
                    .db_instance_class(class)
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe cluster error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    })
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())

```

```

        .build()
    });

mock_rds.expect_describe_db_instance().return_once(|name| {
    Ok(DescribeDbInstancesOutput::builder()
        .db_instances(
            DbInstance::builder()
                .db_instance_identifier(name)
                .db_instance_status("Available")
                .build(),
        )
        .build())
});

mock_rds
    .expect_describe_db_cluster_endpoints()
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
            .build())
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let create = scenario.start_cluster_and_instance().await;
    assert!(create.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::resume();
let _ = assertions.await;
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateDBInstance](#)를 참조하십시오.

## DeleteDBCluster

다음 코드 예시에서는 DeleteDBCluster을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
    let mut clean_up_errors: Vec<ScenarioError> = vec![];

    // Delete the instance. rds.DeleteDbInstance.
    let delete_db_instance = self
        .rds
        .delete_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = delete_db_instance {
        let identifier = self
            .db_instance_identifier
            .as_deref()
            .unwrap_or("Missing Instance Identifier");
        let message = format!("failed to delete db instance {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance to delete
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_instances = self.rds.describe_db_instances().await;
            if let Err(err) = describe_db_instances {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check instance state during deletion",
                    &err,
                ));
                break;
            }
        }
    }
}
```



```

        let db_instances = describe_db_instances
            .unwrap()
            .db_instances()
            .iter()
            .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
            .cloned()
            .collect:::<Vec<DbInstance>>();

        if db_instances.is_empty() {
            trace!("Delete Instance waited and no instances were found");
            break;
        }
        match db_instances.first().unwrap().db_instance_status() {
            Some("Deleting") => continue,
            Some(status) => {
                info!("Attempting to delete but instances is in {status}");
                continue;
            }
            None => {
                warn!("No status for DB instance");
                break;
            }
        }
    }
}

// Delete the DB cluster. rds.DeleteDbCluster.
let delete_db_cluster = self
    .rds
    .delete_db_cluster(
        self.db_cluster_identifier
            .as_deref()
            .expect("cluster identifier"),
    )
    .await;

if let Err(err) = delete_db_cluster {
    let identifier = self
        .db_cluster_identifier
        .as_deref()
        .unwrap_or("Missing DB Cluster Identifier");
    let message = format!("failed to delete db cluster {identifier}");
    clean_up_errors.push(ScenarioError::new(message, &err));
}

```

```

    } else {
        // Wait for the instance and cluster to fully delete.
        rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_clusters = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
            if let Err(err) = describe_db_clusters {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check cluster state during deletion",
                    &err,
                ));
                break;
            }
            let describe_db_clusters = describe_db_clusters.unwrap();
            let db_clusters = describe_db_clusters.db_clusters();
            if db_clusters.is_empty() {
                trace!("Delete cluster waited and no clusters were found");
                break;
            }
            match db_clusters.first().unwrap().status() {
                Some("Deleting") => continue,
                Some(status) => {
                    info!("Attempting to delete but clusters is in {status}");
                    continue;
                }
                None => {
                    warn!("No status for DB cluster");
                    break;
                }
            }
        }
    }

    // Delete the DB cluster parameter group. rds.DeleteDbClusterParameterGroup.
    let delete_db_cluster_parameter_group = self
        .rds
        .delete_db_cluster_parameter_group(

```

```

        self.db_cluster_parameter_group
            .map(|g| {
                g.db_cluster_parameter_group_name
                    .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
            })
            .as_deref()
            .expect("cluster parameter group name"),
    )
    .await;
if let Err(error) = delete_db_cluster_parameter_group {
    clean_up_errors.push(ScenarioError::new(
        "Failed to delete the db cluster parameter group",
        &error,
    ))
}

if clean_up_errors.is_empty() {
    Ok(())
} else {
    Err(clean_up_errors)
}
}

pub async fn delete_db_cluster(
    &self,
    cluster_identifier: &str,
) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
    self.inner
        .delete_db_cluster()
        .db_cluster_identifier(cluster_identifier)
        .skip_final_snapshot(true)
        .send()
        .await
}

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));
}

```

```
mock_rds
    .expect_describe_db_instances()
    .with()
    .times(1)
    .returning(|| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_cluster_identifier("MockCluster")
                    .db_instance_status("Deleting")
                    .build(),
            )
            .build())
    })
    .with()
    .times(1)
    .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

mock_rds
    .expect_delete_db_cluster()
    .with(eq("MockCluster"))
    .return_once(|_| Ok>DeleteDbClusterOutput::builder().build()));

mock_rds
    .expect_describe_db_clusters()
    .with(eq("MockCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(
                DbCluster::builder()
                    .db_cluster_identifier(id)
                    .status("Deleting")
                    .build(),
            )
            .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

mock_rds
    .expect_delete_db_cluster_parameter_group()
```

```

        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
tokio::time::resume();
let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)

```

```

        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

mock_rds
    .expect_delete_db_cluster()
    .with(eq("MockCluster"))
    .return_once(|_| Ok>DeleteDbClusterOutput::builder().build()));

mock_rds
    .expect_describe_db_clusters()
    .with(eq("MockCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(
                DbCluster::builder()
                    .db_cluster_identifier(id)
                    .status("Deleting")
                    .build(),
            )
            .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| {

```

```

        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe db clusters error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster

```

```

    tokio::time::resume();
    let _ = assertions.await;
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteDBCluster](#)를 참조하십시오.

## DeleteDBClusterParameterGroup

다음 코드 예시에서는 DeleteDBClusterParameterGroup을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
    let mut clean_up_errors: Vec<ScenarioError> = vec![];

    // Delete the instance. rds.DeleteDbInstance.
    let delete_db_instance = self
        .rds
        .delete_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = delete_db_instance {
        let identifier = self
            .db_instance_identifier
            .as_deref()
            .unwrap_or("Missing Instance Identifier");
        let message = format!("failed to delete db instance {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance to delete
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {

```



```

        let describe_db_instances = self.rds.describe_db_instances().await;
        if let Err(err) = describe_db_instances {
            clean_up_errors.push(ScenarioError::new(
                "Failed to check instance state during deletion",
                &err,
            ));
            break;
        }
        let db_instances = describe_db_instances
            .unwrap()
            .db_instances()
            .iter()
            .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
            .cloned()
            .collect:::<Vec<DbInstance>>();

        if db_instances.is_empty() {
            trace!("Delete Instance waited and no instances were found");
            break;
        }
        match db_instances.first().unwrap().db_instance_status() {
            Some("Deleting") => continue,
            Some(status) => {
                info!("Attempting to delete but instances is in {status}");
                continue;
            }
            None => {
                warn!("No status for DB instance");
                break;
            }
        }
    }
}

// Delete the DB cluster. rds.DeleteDbCluster.
let delete_db_cluster = self
    .rds
    .delete_db_cluster(
        self.db_cluster_identifier
            .as_deref()
            .expect("cluster identifier"),
    )
    .await;

```

```

    if let Err(err) = delete_db_cluster {
        let identifier = self
            .db_cluster_identifier
            .as_deref()
            .unwrap_or("Missing DB Cluster Identifier");
        let message = format!("failed to delete db cluster {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance and cluster to fully delete.
        rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_clusters = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
            if let Err(err) = describe_db_clusters {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check cluster state during deletion",
                    &err,
                ));
                break;
            }
            let describe_db_clusters = describe_db_clusters.unwrap();
            let db_clusters = describe_db_clusters.db_clusters();
            if db_clusters.is_empty() {
                trace!("Delete cluster waited and no clusters were found");
                break;
            }
            match db_clusters.first().unwrap().status() {
                Some("Deleting") => continue,
                Some(status) => {
                    info!("Attempting to delete but clusters is in {status}");
                    continue;
                }
                None => {
                    warn!("No status for DB cluster");
                    break;
                }
            }
        }
    }
}

```

```

        }
    }
}

// Delete the DB cluster parameter group. rds.DeleteDbClusterParameterGroup.
let delete_db_cluster_parameter_group = self
    .rds
    .delete_db_cluster_parameter_group(
        self.db_cluster_parameter_group
            .map(|g| {
                g.db_cluster_parameter_group_name
                    .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
            })
            .as_deref()
            .expect("cluster parameter group name"),
    )
    .await;
if let Err(error) = delete_db_cluster_parameter_group {
    clean_up_errors.push(ScenarioError::new(
        "Failed to delete the db cluster parameter group",
        &error,
    ))
}

if clean_up_errors.is_empty() {
    Ok(())
} else {
    Err(clean_up_errors)
}
}

pub async fn delete_db_cluster_parameter_group(
    &self,
    name: &str,
) -> Result<DeleteDbClusterParameterGroupOutput,
SdkError<DeleteDBClusterParameterGroupError>>
{
    self.inner
        .delete_db_cluster_parameter_group()
        .db_cluster_parameter_group_name(name)
        .send()
        .await
}

```

```
#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
            )
        })
}
```

```

        )
        .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))
    .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
tokio::time::resume();
let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

```

```
mock_rds
    .expect_delete_db_instance()
    .with(eq("MockInstance"))
    .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

mock_rds
    .expect_describe_db_instances()
    .with()
    .times(1)
    .returning(|| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_cluster_identifier("MockCluster")
                    .db_instance_status("Deleting")
                    .build(),
            )
            .build())
    })
    .with()
    .times(1)
    .returning(|| {
        Err(SdkError::service_error(
            DescribeDBInstancesError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe db instances error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

mock_rds
    .expect_delete_db_cluster()
    .with(eq("MockCluster"))
    .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

mock_rds
    .expect_describe_db_clusters()
    .with(eq("MockCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(
                DbCluster::builder()

```

```

        .db_cluster_identifier(id)
        .status("Deleting")
        .build(),
    )
    .build()
})
.with(eq("MockCluster"))
.times(1)
.returning(|_| {
    Err(SdkError::service_error(
        DescribeDBClustersError::unhandled(Box::new(Error::new(
            ErrorKind::Other,
            "describe db clusters error",
        ))),
        Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
    ))
});

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))
    .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_err());
    let errs = clean_up.unwrap_err();
    assert_eq!(errs.len(), 2);
    assert_matches!(errs.first(), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
    assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
});

```

```

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteDBClusterParameterGroup](#)을 참조하십시오.

## DeleteDBInstance

다음 코드 예시에서는 DeleteDBInstance을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
    let mut clean_up_errors: Vec<ScenarioError> = vec![];

    // Delete the instance. rds.DeleteDbInstance.
    let delete_db_instance = self
        .rds
        .delete_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
}

```



```

if let Err(err) = delete_db_instance {
    let identifier = self
        .db_instance_identifier
        .as_deref()
        .unwrap_or("Missing Instance Identifier");
    let message = format!("failed to delete db instance {identifier}");
    clean_up_errors.push(ScenarioError::new(message, &err));
} else {
    // Wait for the instance to delete
    let waiter = Waiter::default();
    while waiter.sleep().await.is_ok() {
        let describe_db_instances = self.rds.describe_db_instances().await;
        if let Err(err) = describe_db_instances {
            clean_up_errors.push(ScenarioError::new(
                "Failed to check instance state during deletion",
                &err,
            ));
            break;
        }
        let db_instances = describe_db_instances
            .unwrap()
            .db_instances()
            .iter()
            .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
            .cloned()
            .collect:::<Vec<DbInstance>>();

        if db_instances.is_empty() {
            trace!("Delete Instance waited and no instances were found");
            break;
        }
        match db_instances.first().unwrap().db_instance_status() {
            Some("Deleting") => continue,
            Some(status) => {
                info!("Attempting to delete but instances is in {status}");
                continue;
            }
            None => {
                warn!("No status for DB instance");
                break;
            }
        }
    }
}

```

```
    }

    // Delete the DB cluster. rds.DeleteDbCluster.
    let delete_db_cluster = self
        .rds
        .delete_db_cluster(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = delete_db_cluster {
        let identifier = self
            .db_cluster_identifier
            .as_deref()
            .unwrap_or("Missing DB Cluster Identifier");
        let message = format!("failed to delete db cluster {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance and cluster to fully delete.
        rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_clusters = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
            if let Err(err) = describe_db_clusters {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check cluster state during deletion",
                    &err,
                ));
                break;
            }
            let describe_db_clusters = describe_db_clusters.unwrap();
            let db_clusters = describe_db_clusters.db_clusters();
            if db_clusters.is_empty() {
                trace!("Delete cluster waited and no clusters were found");
                break;
            }
        }
    }
}
```

```

        }
        match db_clusters.first().unwrap().status() {
            Some("Deleting") => continue,
            Some(status) => {
                info!("Attempting to delete but clusters is in {status}");
                continue;
            }
            None => {
                warn!("No status for DB cluster");
                break;
            }
        }
    }
}

// Delete the DB cluster parameter group. rds.DeleteDbClusterParameterGroup.
let delete_db_cluster_parameter_group = self
    .rds
    .delete_db_cluster_parameter_group(
        self.db_cluster_parameter_group
            .map(|g| {
                g.db_cluster_parameter_group_name
                    .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
            })
            .as_deref()
            .expect("cluster parameter group name"),
    )
    .await;
if let Err(error) = delete_db_cluster_parameter_group {
    clean_up_errors.push(ScenarioError::new(
        "Failed to delete the db cluster parameter group",
        &error,
    ))
}

if clean_up_errors.is_empty() {
    Ok(())
} else {
    Err(clean_up_errors)
}
}

pub async fn delete_db_instance(

```

```

        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }
}

```

```
#[tokio::test]
```

```

async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds

```

```

    .expect_describe_db_clusters()
    .with(eq("MockCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(
                DbCluster::builder()
                    .db_cluster_identifier(id)
                    .status("Deleting")
                    .build(),
            )
            .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))
    .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster

```

```

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))

```

```

        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

mock_rds
    .expect_describe_db_clusters()
    .with(eq("MockCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(
                DbCluster::builder()
                    .db_cluster_identifier(id)
                    .status("Deleting")
                    .build(),
            )
            .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe db clusters error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))
    .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();

```

```

let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_err());
    let errs = clean_up.unwrap_err();
    assert_eq!(errs.len(), 2);
    assert_matches!(errs.first(), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
    assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
});

tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
tokio::time::resume();
let _ = assertions.await;
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteDBInstance](#)를 참조하십시오.

## DescribeDBClusterParameters

다음 코드 예시에서는 DescribeDBClusterParameters을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

// Get the parameter group. rds.DescribeDbClusterParameterGroups
// Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters

```



```

    // Parse the ParameterName, Description, and AllowedValues values and display
    them.
    pub async fn cluster_parameters(&self) -> Result<Vec<AuroraScenarioParameter>,
ScenarioError> {
        let parameters_output = self
            .rds
            .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
            .await;

        if let Err(err) = parameters_output {
            return Err(ScenarioError::new(
                format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
                &err,
            ));
        }

        let parameters = parameters_output
            .unwrap()
            .into_iter()
            .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
            .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
            .map(AuroraScenarioParameter::from)
            .collect::<Vec<_>>();

        Ok(parameters)
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }
}

```

```

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))

```

```

        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
let params = scenario.cluster_parameters().await;
assert_matches!(params, Err(ScenarioError { message, context: _ }) if message ==
"Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeDBClusterParameters](#)를 참조하십시오.

## DescribeDBClusters

다음 코드 예시에서는 DescribeDBClusters을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

// Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
// Create an Aurora DB cluster database cluster that contains a MySQL database
and uses the parameter group you created.
// Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
// Get a list of instance classes available for the selected engine and engine
version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql', EngineVersion=).

```

```
// Create a database instance in the cluster.
// Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
DBInstanceStatus == 'available'.
pub async fn start_cluster_and_instance(&mut self) -> Result<(), ScenarioError>
{
    if self.password.is_none() {
        return Err(ScenarioError::with(
            "Must set Secret Password before starting a cluster",
        ));
    }
    let create_db_cluster = self
        .rds
        .create_db_cluster(
            DB_CLUSTER_IDENTIFIER,
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            DB_ENGINE,
            self.engine_version.as_deref().expect("engine version"),
            self.username.as_deref().expect("username"),
            self.password
                .replace(SecretString::new("").to_string())
                .expect("password"),
        )
        .await;
    if let Err(err) = create_db_cluster {
        return Err(ScenarioError::new(
            "Failed to create DB Cluster with cluster group",
            &err,
        ));
    }

    self.db_cluster_identifier = create_db_cluster
        .unwrap()
        .db_cluster
        .and_then(|c| c.db_cluster_identifier);

    if self.db_cluster_identifier.is_none() {
        return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
    }

    info!(
        "Started a db cluster: {}",
        self.db_cluster_identifier
            .as_deref()
    );
}
```

```
        .unwrap_or("Missing ARN")
    );

    let create_db_instance = self
        .rds
        .create_db_instance(
            self.db_cluster_identifier.as_deref().expect("cluster name"),
            DB_INSTANCE_IDENTIFIER,
            self.instance_class.as_deref().expect("instance class"),
            DB_ENGINE,
        )
        .await;
    if let Err(err) = create_db_instance {
        return Err(ScenarioError::new(
            "Failed to create Instance in DB Cluster",
            &err,
        ));
    }

    self.db_instance_identifier = create_db_instance
        .unwrap()
        .db_instance
        .and_then(|i| i.db_instance_identifier);

    // Cluster creation can take up to 20 minutes to become available
    let cluster_max_wait = Duration::from_secs(20 * 60);
    let waiter = Waiter::builder().max(cluster_max_wait).build();
    while waiter.sleep().await.is_ok() {
        let cluster = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = cluster {
            warn!(?err, "Failed to describe cluster while waiting for ready");
            continue;
        }
    }

    let instance = self
        .rds
```

```
        .describe_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = instance {
        return Err(ScenarioError::new(
            "Failed to find instance for cluster",
            &err,
        ));
    }

    let instances_available = instance
        .unwrap()
        .db_instances()
        .iter()
        .all(|instance| instance.db_instance_status() == Some("Available"));

    let endpoints = self
        .rds
        .describe_db_cluster_endpoints(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = endpoints {
        return Err(ScenarioError::new(
            "Failed to find endpoint for cluster",
            &err,
        ));
    }

    let endpoints_available = endpoints
        .unwrap()
        .db_cluster_endpoints()
        .iter()
        .all(|endpoint| endpoint.status() == Some("available"));

    if instances_available && endpoints_available {
        return Ok(());
    }
}
```

```

    }

    Err(ScenarioError::with("timed out waiting for cluster"))
}

pub async fn describe_db_clusters(
    &self,
    id: &str,
) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
    self.inner
        .describe_db_clusters()
        .db_cluster_identifier(id)
        .send()
        .await
}

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
        });
}

```

```

        true
    })
    .return_once(|cluster, name, class, _| {
        Ok(CreateDbInstanceOutput::builder()
            .db_instance(
                DbInstance::builder()
                    .db_cluster_identifier(cluster)
                    .db_instance_identifier(name)
                    .db_instance_class(class)
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_describe_db_instance()
    .with(eq("RustSDKCodeExamplesDBInstance"))
    .return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_cluster_endpoints()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
    });

```



```

        .build()
    });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });
}

```

```

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

let create = scenario.start_cluster_and_instance().await;
assert_matches!(create, Err(ScenarioError { message, context: _}) if message ==
"Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build());
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
        });
}

```

```

        assert_eq!(engine, "aurora-mysql");
        assert_eq!(version, "aurora-mysql8.0");
        assert_eq!(username, "test username");
        assert_eq!(password.expose_secret(), "test password");
        true
    })
    .return_once(|id, _, _, _, _| {
        Ok(CreateDbClusterOutput::builder()
            .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_create_db_instance()
    .return_once(|_, _, _, _| {
        Err(SdkError::service_error(
            CreateDBInstanceError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "create db instance error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

let create = scenario.start_cluster_and_instance().await;
assert_matches!(create, Err(ScenarioError { message, context: _ }) if message ==
"Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
        });

```

```

        assert_eq!(engine, "aurora-mysql");
        assert_eq!(version, "aurora-mysql8.0");
        assert_eq!(username, "test username");
        assert_eq!(password.expose_secret(), "test password");
        true
    })
    .return_once(|id, _, _, _, _, _| {
        Ok(CreateDbClusterOutput::builder()
            .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds
    .expect_create_db_instance()
    .withf(|cluster, name, class, engine| {
        assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
        assert_eq!(name, "RustSDKCodeExamplesDBInstance");
        assert_eq!(class, "m5.large");
        assert_eq!(engine, "aurora-mysql");
        true
    })
    .return_once(|cluster, name, class, _| {
        Ok(CreateDbInstanceOutput::builder()
            .db_instance(
                DbInstance::builder()
                    .db_cluster_identifier(cluster)
                    .db_instance_identifier(name)
                    .db_instance_class(class)
                    .build(),
            )
            .build())
    });

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe cluster error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

```

```

        ))
    })
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
            .build())
    });

mock_rds.expect_describe_db_instance().return_once(|name| {
    Ok(DescribeDbInstancesOutput::builder()
        .db_instances(
            DbInstance::builder()
                .db_instance_identifier(name)
                .db_instance_status("Available")
                .build(),
        )
        .build())
});

mock_rds
    .expect_describe_db_cluster_endpoints()
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
            .build())
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_version = Some("aurora-mysql8.0".into());
scenario.instance_class = Some("m5.large".into());
scenario.username = Some("test username".into());
scenario.password = Some(SecretString::new("test password".into()));

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let create = scenario.start_cluster_and_instance().await;
    assert!(create.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::advance(Duration::from_secs(1)).await;

```

```

    tokio::time::resume();
    let _ = assertions.await;
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeDBClusters](#)를 참조하십시오.

## DescribeDBEngineVersions

다음 코드 예시에서는 DescribeDBEngineVersions을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

// Get available engine families for Aurora MySQL.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
    let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
    trace!(versions=?describe_db_engine_versions, "full list of versions");

    if let Err(err) = describe_db_engine_versions {
        return Err(ScenarioError::new(
            "Failed to retrieve DB Engine Versions",
            &err,
        ));
    };

    let version_count = describe_db_engine_versions
        .as_ref()
        .map(|o| o.db_engine_versions().len())
        .unwrap_or_default();
    info!(version_count, "got list of versions");

    // Create a map of engine families to their available versions.

```

```

let mut versions = HashMap::

```

```

        )
        .db_engine_versions(
            DbEngineVersion::builder()
                .db_parameter_group_family("f1")
                .engine_version("f1b")
                .build(),
        )
        .db_engine_versions(
            DbEngineVersion::builder()
                .db_parameter_group_family("f2")
                .engine_version("f2a")
                .build(),
        )
        .db_engine_versions(DbEngineVersion::builder().build())
        .build()
    });

let scenario = AuroraScenario::new(mock_rds);

let versions_map = scenario.get_engines().await;

assert_eq!(
    versions_map,
    Ok(HashMap::from([
        ("f1".into(), vec!["f1a".into(), "f1b".into()]),
        ("f2".into(), vec!["f2a".into()])
    ]))
);
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
            )),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        )
    }

```



```

        ))
    });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to retrieve
DB Engine Versions"
    );
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeDBEngineVersions](#)을 참조하십시오.

## DescribeDBInstances

다음 코드 예시에서는 DescribeDBInstances을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
    let mut clean_up_errors: Vec<ScenarioError> = vec![];

    // Delete the instance. rds.DeleteDbInstance.
    let delete_db_instance = self
        .rds
        .delete_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = delete_db_instance {
        let identifier = self

```

```

        .db_instance_identifier
        .as_deref()
        .unwrap_or("Missing Instance Identifier");
    let message = format!("failed to delete db instance {identifier}");
    clean_up_errors.push(ScenarioError::new(message, &err));
} else {
    // Wait for the instance to delete
    let waiter = Waiter::default();
    while waiter.sleep().await.is_ok() {
        let describe_db_instances = self.rds.describe_db_instances().await;
        if let Err(err) = describe_db_instances {
            clean_up_errors.push(ScenarioError::new(
                "Failed to check instance state during deletion",
                &err,
            ));
            break;
        }
        let db_instances = describe_db_instances
            .unwrap()
            .db_instances()
            .iter()
            .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
            .cloned()
            .collect:::<Vec<DbInstance>>();

        if db_instances.is_empty() {
            trace!("Delete Instance waited and no instances were found");
            break;
        }
        match db_instances.first().unwrap().db_instance_status() {
            Some("Deleting") => continue,
            Some(status) => {
                info!("Attempting to delete but instances is in {status}");
                continue;
            }
            None => {
                warn!("No status for DB instance");
                break;
            }
        }
    }
}
}

```

```

// Delete the DB cluster. rds.DeleteDbCluster.
let delete_db_cluster = self
    .rds
    .delete_db_cluster(
        self.db_cluster_identifier
            .as_deref()
            .expect("cluster identifier"),
    )
    .await;

if let Err(err) = delete_db_cluster {
    let identifier = self
        .db_cluster_identifier
        .as_deref()
        .unwrap_or("Missing DB Cluster Identifier");
    let message = format!("failed to delete db cluster {identifier}");
    clean_up_errors.push(ScenarioError::new(message, &err));
} else {
    // Wait for the instance and cluster to fully delete.
    rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
    let waiter = Waiter::default();
    while waiter.sleep().await.is_ok() {
        let describe_db_clusters = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;
        if let Err(err) = describe_db_clusters {
            clean_up_errors.push(ScenarioError::new(
                "Failed to check cluster state during deletion",
                &err,
            ));
            break;
        }
        let describe_db_clusters = describe_db_clusters.unwrap();
        let db_clusters = describe_db_clusters.db_clusters();
        if db_clusters.is_empty() {
            trace!("Delete cluster waited and no clusters were found");
            break;
        }
        match db_clusters.first().unwrap().status() {

```

```

        Some("Deleting") => continue,
        Some(status) => {
            info!("Attempting to delete but clusters is in {status}");
            continue;
        }
        None => {
            warn!("No status for DB cluster");
            break;
        }
    }
}

// Delete the DB cluster parameter group. rds.DeleteDbClusterParameterGroup.
let delete_db_cluster_parameter_group = self
    .rds
    .delete_db_cluster_parameter_group(
        self.db_cluster_parameter_group
            .map(|g| {
                g.db_cluster_parameter_group_name
                    .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
            })
            .as_deref()
            .expect("cluster parameter group name"),
    )
    .await;
if let Err(error) = delete_db_cluster_parameter_group {
    clean_up_errors.push(ScenarioError::new(
        "Failed to delete the db cluster parameter group",
        &error,
    ))
}

if clean_up_errors.is_empty() {
    Ok(())
} else {
    Err(clean_up_errors)
}
}

pub async fn describe_db_instances(
    &self,
) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {

```

```
        self.inner.describe_db_instances().send().await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)

```

```

                .status("Deleting")
                .build(),
            )
            .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))
    .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_ok());
});

tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
tokio::time::resume();
let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {

```

```
let mut mock_rds = MockRdsImpl::default();

mock_rds
    .expect_delete_db_instance()
    .with(eq("MockInstance"))
    .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

mock_rds
    .expect_describe_db_instances()
    .with()
    .times(1)
    .returning(|| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_cluster_identifier("MockCluster")
                    .db_instance_status("Deleting")
                    .build(),
            )
            .build())
    })
    .with()
    .times(1)
    .returning(|| {
        Err(SdkError::service_error(
            DescribeDBInstancesError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe db instances error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

mock_rds
    .expect_delete_db_cluster()
    .with(eq("MockCluster"))
    .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

mock_rds
    .expect_describe_db_clusters()
    .with(eq("MockCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
```

```

        .db_clusters(
            DbCluster::builder()
                .db_cluster_identifier(id)
                .status("Deleting")
                .build(),
        )
        .build()
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "describe db clusters error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))
    .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_err());
    let errs = clean_up.unwrap_err();
    assert_eq!(errs.len(), 2);
    assert_matches!(errs.first(), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
});

```



```

        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeDBInstances](#)를 참조하십시오.

## DescribeOrderableDBInstanceOptions

다음 코드 예시에서는 DescribeOrderableDBInstanceOptions을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn get_instance_classes(&self) -> Result<Vec<String>, ScenarioError> {
    let describe_orderable_db_instance_options_items = self
        .rds
        .describe_orderable_db_instance_options(
            DB_ENGINE,
            self.engine_version
                .as_ref()
                .expect("engine version for db instance options")
                .as_str(),
        )
        .await;
}

```

```

describe_orderable_db_instance_options_items
    .map(|options| {
        options
            .iter()
            .filter(|o| o.storage_type() == Some("aurora"))
            .map(|o| o.db_instance_class().unwrap_or_default().to_string())
            .collect::<Vec<String>>()
    })
    .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
}

pub async fn describe_orderable_db_instance_options(
    &self,
    engine: &str,
    engine_version: &str,
) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
{
    self.inner
        .describe_orderable_db_instance_options()
        .engine(engine)
        .engine_version(engine_version)
        .into_paginator()
        .items()
        .send()
        .try_collect()
        .await
}

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

.db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });
}

```

```

mock_rds
    .expect_describe_orderable_db_instance_options()
    .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
    .return_once(|_, _| {
        Ok(vec![
            OrderableDbInstanceOption::builder()
                .db_instance_class("t1")
                .storage_type("aurora")
                .build(),
            OrderableDbInstanceOption::builder()
                .db_instance_class("t1")
                .storage_type("aurora-iopt1")
                .build(),
            OrderableDbInstanceOption::builder()
                .db_instance_class("t2")
                .storage_type("aurora")
                .build(),
            OrderableDbInstanceOption::builder()
                .db_instance_class("t3")
                .storage_type("aurora")
                .build(),
        ])
    });

let mut scenario = AuroraScenario::new(mock_rds);
scenario
    .set_engine("aurora-mysql", "aurora-mysql8.0")
    .await
    .expect("set engine");

let instance_classes = scenario.get_instance_classes().await;

assert_eq!(
    instance_classes,
    Ok(vec!["t1".into(), "t2".into(), "t3".into()])
);
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()

```

```

        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(
                DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

let mut scenario = AuroraScenario::new(mock_rds);
scenario.engine_family = Some("aurora-mysql".into());
scenario.engine_version = Some("aurora-mysql8.0".into());

let instance_classes = scenario.get_instance_classes().await;

assert_matches!(
    instance_classes,
    Err(ScenarioError {message, context: _}) if message == "Could not get
available instance classes"
);
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeOrderableDBInstanceOptions](#)를 참조하십시오.

## ModifyDBClusterParameterGroup

다음 코드 예시에서는 ModifyDBClusterParameterGroup을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

// Modify both the auto_increment_offset and auto_increment_increment parameters
in one call in the custom parameter group. Set their ParameterValue fields to a new
allowable value. rds.ModifyDbClusterParameterGroup.
pub async fn update_auto_increment(
    &self,
    offset: u8,
    increment: u8,
) -> Result<(), ScenarioError> {
    let modify_db_cluster_parameter_group = self
        .rds
        .modify_db_cluster_parameter_group(
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            vec![
                Parameter::builder()
                    .parameter_name("auto_increment_offset")
                    .parameter_value(format!("{offset}"))
                    .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                    .build(),
                Parameter::builder()
                    .parameter_name("auto_increment_increment")
                    .parameter_value(format!("{increment}"))
                    .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                    .build(),
            ],
        )
        .await;

    if let Err(error) = modify_db_cluster_parameter_group {
        return Err(ScenarioError::new(
            "Failed to modify cluster parameter group",
            &error,
        ));
    }

    Ok(())
}

pub async fn modify_db_cluster_parameter_group(
    &self,
    name: &str,
    parameters: Vec<Parameter>,
) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>

```

```

    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

```

```

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(
                ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message ==
"Failed to modify cluster parameter group");
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ModifyDBClusterParameterGroup](#)을 참조하십시오.

## SDK for Rust를 사용한 Auto Scaling 예제

다음 코드 예제에서는 Auto Scaling과 함께 AWS SDK for Rust를 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.

## 시작

### Auto Scaling 시작

다음 코드 예제에서는 Auto Scaling을 사용하여 시작하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
async fn list_groups(client: &Client) -> Result<(), Error> {
    let resp = client.describe_auto_scaling_groups().send().await?;

    println!("Groups:");

    let groups = resp.auto_scaling_groups();

    for group in groups {
        println!(
            "Name: {}",
            group.auto_scaling_group_name().unwrap_or("Unknown")
        );
        println!(
            "Arn:  {}",
            group.auto_scaling_group_arn().unwrap_or("unknown"),
        );
        println!("Zones: {:?}", group.availability_zones(),);
        println!();
    }

    println!("Found {} group(s)", groups.len());

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeAutoScalingGroups](#)을 참조하십시오.



## 주제

- [기본 사항](#)
- [작업](#)

## 기본 사항

### 기본 사항 알아보기

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 시작 템플릿과 가용 영역이 있는 Amazon EC2 Auto Scaling 그룹을 생성하고 실행 중인 인스턴스에 대한 정보를 가져옵니다.
- Amazon CloudWatch 지표 수집 활성화
- 그룹의 원하는 용량을 업데이트하고 인스턴스가 시작될 때까지 기다립니다.
- 그룹에서 인스턴스를 종료합니다.
- 사용자 요청 및 용량 변경에 따라 발생하는 조정 활동을 나열합니다.
- CloudWatch 지표에 대한 통계를 가져온 다음 리소스를 정리합니다.

### SDK for Rust

#### Note

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

```
[package]
name = "autoscaling-code-examples"
version = "0.1.0"
authors = ["Doug Schwartz <dougsch@amazon.com>", "David Souther
<dpsouth@amazon.com>"]
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/
manifest.html

[dependencies]
```

```

aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-autoscaling = { version = "1.3.0" }
aws-sdk-ec2 = { version = "1.3.0" }
aws-types = { version = "1.0.1" }
tokio = { version = "1.20.1", features = ["full"] }
clap = { version = "4.4", features = ["derive"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
anyhow = "1.0.75"
tracing = "0.1.37"
tokio-stream = "0.1.14"

use std::{collections::BTreeSet, fmt::Display};

use anyhow::anyhow;
use autoscaling_code_examples::scenario::{AutoScalingScenario, ScenarioError};
use tracing::{info, warn};

async fn show_scenario_description(scenario: &AutoScalingScenario, event: &str) {
    let description = scenario.describe_scenario().await;
    info!("DescribeAutoScalingInstances: {event}\n{description}");
}

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    pub fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    pub fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:");
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "");
        }
    }
}

```

```
        Ok(())
    }
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();

    let shared_config = aws_config::from_env().load().await;

    let mut warnings = Warnings::default();

    // 1. Create an EC2 launch template that you'll use to create an auto scaling
    // group. Bonus: use SDK with EC2.CreateLaunchTemplate to create the launch template.
    // 2. CreateAutoScalingGroup: pass it the launch template you created in step 0.
    // Give it min/max of 1 instance.
    // 4. EnableMetricsCollection: enable all metrics or a subset.
    let scenario = match AutoScalingScenario::prepare_scenario(&shared_config).await
    {
        Ok(scenario) => scenario,
        Err(errs) => {
            let err_str = errs
                .into_iter()
                .map(|e| e.to_string())
                .collect::<Vec<String>>()
                .join(", ");
            return Err(anyhow!("Failed to initialize scenario: {err_str}"));
        }
    };

    info!("Prepared autoscaling scenario:\n{scenario}");

    let stable = scenario.wait_for_stable(1).await;
    if let Err(err) = stable {
        warnings.push(
            "There was a problem while waiting for group to be stable",
            err,
        );
    }

    // 3. DescribeAutoScalingInstances: show that one instance has launched.
    show_scenario_description(
        &scenario,
        "show that the group was created and one instance has launched",
    );
}
```

```
)
.await;

// 5. UpdateAutoScalingGroup: update max size to 3.
let scale_max_size = scenario.scale_max_size(3).await;
if let Err(err) = scale_max_size {
    warnings.push("There was a problem scaling max size", err);
}

// 6. DescribeAutoScalingGroups: the current state of the group
show_scenario_description(
    &scenario,
    "show the current state of the group after setting max size",
)
.await;

// 7. SetDesiredCapacity: set desired capacity to 2.
let scale_desired_capacity = scenario.scale_desired_capacity(2).await;
if let Err(err) = scale_desired_capacity {
    warnings.push("There was a problem setting desired capacity", err);
}

// Wait for a second instance to launch.
let stable = scenario.wait_for_stable(2).await;
if let Err(err) = stable {
    warnings.push(
        "There was a problem while waiting for group to be stable",
        err,
    );
}

// 8. DescribeAutoScalingInstances: show that two instances are launched.
show_scenario_description(
    &scenario,
    "show that two instances are launched after setting desired capacity",
)
.await;

let ids_before = scenario
    .list_instances()
    .await
    .map(|v| v.into_iter().collect::<BTreeSet<_>>())
    .unwrap_or_default();
```

```
// 9. TerminateInstanceInAutoScalingGroup: terminate one of the instances in the
group.
let terminate_some_instance = scenario.terminate_some_instance().await;
if let Err(err) = terminate_some_instance {
    warnings.push("There was a problem replacing an instance", err);
}

let wait_after_terminate = scenario.wait_for_stable(1).await;
if let Err(err) = wait_after_terminate {
    warnings.push(
        "There was a problem waiting after terminating an instance",
        err,
    );
}

let wait_scale_up_after_terminate = scenario.wait_for_stable(2).await;
if let Err(err) = wait_scale_up_after_terminate {
    warnings.push(
        "There was a problem waiting for scale up after terminating an
instance",
        err,
    );
}

let ids_after = scenario
    .list_instances()
    .await
    .map(|v| v.into_iter().collect:::<BTreeSet<_>>())
    .unwrap_or_default();

let difference = ids_after.intersection(&ids_before).count();
if !(difference == 1 && ids_before.len() == 2 && ids_after.len() == 2) {
    warnings.push(
        "Before and after set not different",
        ScenarioError::with(format!("{}", difference)),
    );
}

// 10. DescribeScalingActivities: list the scaling activities that have occurred
for the group so far.
show_scenario_description(
    &scenario,
    "list the scaling activities that have occurred for the group so far",
)
```

```
.await;

// 11. DisableMetricsCollection
let scale_group = scenario.scale_group_to_zero().await;
if let Err(err) = scale_group {
    warnings.push("There was a problem scaling the group to 0", err);
}
show_scenario_description(&scenario, "Scenario scaled to 0").await;

// 12. DeleteAutoScalingGroup (to delete the group you must stop all instances):
// 13. Delete LaunchTemplate.
let clean_scenario = scenario.clean_scenario().await;
if let Err(errs) = clean_scenario {
    for err in errs {
        warnings.push("There was a problem cleaning the scenario", err);
    }
} else {
    info!("The scenario has been cleaned up!");
}

if warnings.is_empty() {
    Ok(())
} else {
    Err(anyhow!(
        "There were warnings during scenario execution:\n{warnings}"
    ))
}
}

pub mod scenario;

use std::{
    error::Error,
    fmt::{Debug, Display},
    time::{Duration, SystemTime},
};

use anyhow::anyhow;
use aws_config::SdkConfig;
use aws_sdk_autoscaling::{
    error::{DisplayErrorContext, ProvideErrorMetadata},
    types::{Activity, AutoScalingGroup, LaunchTemplateSpecification},
};
```

```
use aws_sdk_ec2::types::RequestLaunchTemplateData;
use tracing::trace;

const LAUNCH_TEMPLATE_NAME: &str =
    "SDK_Code_Examples_EC2_Autoscaling_template_from_Rust_SDK";
const AUTOSCALING_GROUP_NAME: &str =
    "SDK_Code_Examples_EC2_Autoscaling_Group_from_Rust_SDK";
const MAX_WAIT: Duration = Duration::from_secs(5 * 60); // Wait at most 25 seconds.
const WAIT_TIME: Duration = Duration::from_millis(500); // Wait half a second at a
    time.

struct Waiter {
    start: SystemTime,
    max: Duration,
}

impl Waiter {
    fn new() -> Self {
        Waiter {
            start: SystemTime::now(),
            max: MAX_WAIT,
        }
    }

    async fn sleep(&self) -> Result<(), ScenarioError> {
        if SystemTime::now()
            .duration_since(self.start)
            .unwrap_or(Duration::MAX)
            > self.max
        {
            Err(ScenarioError::with(
                "Exceeded maximum wait duration for stable group",
            ))
        } else {
            tokio::time::sleep(WAIT_TIME).await;
            Ok(())
        }
    }
}

pub struct AutoScalingScenario {
    ec2: aws_sdk_ec2::Client,
    autoscaling: aws_sdk_autoscaling::Client,
    launch_template_arn: String,
```

```

    auto_scaling_group_name: String,
}

impl Display for AutoScalingScenario {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        f.write_fmt(format_args!(
            "\tLaunch Template ID: {}\n",
            self.launch_template_arn
        ))?;
        f.write_fmt(format_args!(
            "\tScaling Group Name: {}\n",
            self.auto_scaling_group_name
        ))?;

        Ok(())
    }
}

pub struct AutoScalingScenarioDescription {
    group: Result<Vec<String>, ScenarioError>,
    instances: Result<Vec<String>, anyhow::Error>,
    activities: Result<Vec<Activity>, anyhow::Error>,
}

impl Display for AutoScalingScenarioDescription {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "\t\t\t\t\t Group status:");
        match &self.group {
            Ok(groups) => {
                for status in groups {
                    writeln!(f, "\t\t\t\t\t- {status}")?;
                }
            }
            Err(e) => writeln!(f, "\t\t\t\t\t! - {e}")?,
        }
        writeln!(f, "\t\t\t\t\t Instances:");
        match &self.instances {
            Ok(instances) => {
                for instance in instances {
                    writeln!(f, "\t\t\t\t\t- {instance}")?;
                }
            }
            Err(e) => writeln!(f, "\t\t\t\t\t! {e}")?,
        }
    }
}

```





```

        (Some(message), Some(code)) => format!("{message} ({code})"),
    };
    write!(f, "{display}")
}
}

#[derive(Debug)]
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) -> Self {
        ScenarioError {
            message: message.into(),
            context: Some(MetadataError::from(err)),
        }
    }
}

impl Error for ScenarioError {
    // While `Error` can capture `source` information about the underlying error,
    // for this example
    // the ScenarioError captures the underlying information in MetadataError and
    // treats it as a
    // single Error from this Crate. In other contexts, it may be appropriate to
    // model the error
    // as including the SdkError as its source.
}

impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}
}

```

```

}

impl AutoScalingScenario {
    pub async fn prepare_scenario(sdk_config: &SdkConfig) -> Result<Self,
Vec<ScenarioError>> {
        let ec2 = aws_sdk_ec2::Client::new(sdk_config);
        let autoscaling = aws_sdk_autoscaling::Client::new(sdk_config);

        let auto_scaling_group_name = String::from(AUTOSCALING_GROUP_NAME);

        // Before creating any resources, prepare the list of AZs
        let availability_zones = ec2.describe_availability_zones().send().await;
        if let Err(err) = availability_zones {
            return Err(vec![ScenarioError::new("Failed to find AZs", &err)]);
        }

        let availability_zones: Vec<String> = availability_zones
            .unwrap()
            .availability_zones
            .unwrap_or_default()
            .iter()
            .take(3)
            .map(|z| z.zone_name.clone().unwrap())
            .collect();

        // 1. Create an EC2 launch template that you'll use to create an auto
        scaling group. Bonus: use SDK with EC2.CreateLaunchTemplate to create the launch
        template.
        // * Recommended: InstanceType='t1.micro', ImageId='ami-0ca285d4c2cda3300'
        let create_launch_template = ec2
            .create_launch_template()
            .launch_template_name(LAUNCH_TEMPLATE_NAME)
            .launch_template_data(
                RequestLaunchTemplateData::builder()
                    .instance_type(aws_sdk_ec2::types::InstanceType::T1Micro)
                    .image_id("ami-0ca285d4c2cda3300")
                    .build(),
            )
            .send()
            .await
            .map_err(|err| vec![ScenarioError::new("Failed to create launch
template", &err)])?;

        let launch_template_arn = match create_launch_template.launch_template {

```

```

        Some(launch_template) =>
launch_template.launch_template_id.unwrap_or_default(),
        None => {
            // Try to delete the launch template
            let _ = ec2
                .delete_launch_template()
                .launch_template_name(LAUNCH_TEMPLATE_NAME)
                .send()
                .await;
            return Err(vec![ScenarioError::with("Failed to load launch
template")]);
        }
    };

    // 2. CreateAutoScalingGroup: pass it the launch template you created in
step 0. Give it min/max of 1 instance.
    // You can use EC2.describe_availability_zones() to get a list of AZs (you
have to specify an AZ when you create the group).
    // Wait for instance to launch. Use a waiter if you have one, otherwise
DescribeAutoScalingInstances until LifecycleState='InService'
    if let Err(err) = autoscaling
        .create_auto_scaling_group()
        .auto_scaling_group_name(auto_scaling_group_name.as_str())
        .launch_template(
            LaunchTemplateSpecification::builder()
                .launch_template_id(launch_template_arn.clone())
                .version("$Latest")
                .build(),
        )
        .max_size(1)
        .min_size(1)
        .set_availability_zones(Some(availability_zones))
        .send()
        .await
    {
        let mut errs = vec![ScenarioError::new(
            "Failed to create autoscaling group",
            &err,
        )];

        if let Err(err) = autoscaling
            .delete_auto_scaling_group()
            .auto_scaling_group_name(auto_scaling_group_name.as_str())
            .send()

```

```

        .await
    {
        errs.push(ScenarioError::new(
            "Failed to clean up autoscaling group",
            &err,
        ));
    }

    if let Err(err) = ec2
        .delete_launch_template()
        .launch_template_id(launch_template_arn.clone())
        .send()
        .await
    {
        errs.push(ScenarioError::new(
            "Failed to clean up launch template",
            &err,
        ));
    }
    return Err(errs);
}

let scenario = AutoScalingScenario {
    ec2,
    autoscaling: autoscaling.clone(), // Clients are cheap so cloning here
to prevent a move is ok.
    auto_scaling_group_name: auto_scaling_group_name.clone(),
    launch_template_arn,
};

let enable_metrics_collection = autoscaling
    .enable_metrics_collection()
    .auto_scaling_group_name(auto_scaling_group_name.as_str())
    .granularity("1Minute")
    .set_metrics(Some(vec![
        String::from("GroupMinSize"),
        String::from("GroupMaxSize"),
        String::from("GroupDesiredCapacity"),
        String::from("GroupInServiceInstances"),
        String::from("GroupTotalInstances"),
    ]))
    .send()
    .await;

```

```

    match enable_metrics_collection {
        Ok(_) => Ok(scenario),
        Err(err) => {
            scenario.clean_scenario().await?;
            Err(vec![ScenarioError::new(
                "Failed to enable metrics collections for group",
                &err,
            )])
        }
    }
}

pub async fn clean_scenario(self) -> Result<(), Vec<ScenarioError>> {
    let _ = self.wait_for_no_scaling().await;
    let delete_group = self
        .autoscaling
        .delete_auto_scaling_group()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .send()
        .await;

    // 14. Delete LaunchTemplate.
    let delete_launch_template = self
        .ec2
        .delete_launch_template()
        .launch_template_id(self.launch_template_arn.clone())
        .send()
        .await;

    let early_exit = match (delete_group, delete_launch_template) {
        (Ok(_), Ok(_)) => Ok(()),
        (Ok(_), Err(e)) => Err(vec![ScenarioError::new(
            "There was an error cleaning the launch template",
            &e,
        )]),
        (Err(e), Ok(_)) => Err(vec![ScenarioError::new(
            "There was an error cleaning the scale group",
            &e,
        )]),
        (Err(e1), Err(e2)) => Err(vec![
            ScenarioError::new("Multiple error cleaning the scenario Scale
Group", &e1),
            ScenarioError::new("Multiple error cleaning the scenario Launch
Template", &e2),
        ])
    }
}

```

```

    ]),
};

if early_exit.is_err() {
    early_exit
} else {
    // Wait for delete_group to finish
    let waiter = Waiter::new();
    let mut errors = Vec::<ScenarioError>::new();
    while errors.len() < 3 {
        if let Err(e) = waiter.sleep().await {
            errors.push(e);
            continue;
        }
        let describe_group = self
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await;
        match describe_group {
            Ok(group) => match group.auto_scaling_groups().first() {
                Some(group) => {
                    if group.status() != Some("Delete in progress") {
                        errors.push(ScenarioError::with(format!(
                            "Group in an unknown state while deleting: {}",
                            group.status().unwrap_or("unknown error")
                        )));
                    }
                    return Err(errors);
                }
                None => return Ok(()),
            },
            Err(err) => {
                errors.push(ScenarioError::new("Failed to describe
autoscaling group during cleanup 3 times, last error", &err));
            }
        }
        if errors.len() > 3 {
            return Err(errors);
        }
    }
    Err(vec![ScenarioError::with(

```

```

        "Exited cleanup wait loop without returning success or failing after
three rounds",
    )])
    }
}

pub async fn describe_scenario(&self) -> AutoScalingScenarioDescription {
    let group = self
        .autoscaling
        .describe_auto_scaling_groups()
        .auto_scaling_group_names(self.auto_scaling_group_name.clone())
        .send()
        .await
        .map(|s| {
            s.auto_scaling_groups()
                .iter()
                .map(|s| {
                    format!(
                        "{}: {}",
                        s.auto_scaling_group_name().unwrap_or("Unknown"),
                        s.status().unwrap_or("Unknown")
                    )
                })
                .collect:::<Vec<String>>()
        })
        .map_err(|e| {
            ScenarioError::new("Failed to describe auto scaling groups for
scenario", &e)
        });

    let instances = self
        .list_instances()
        .await
        .map_err(|e| anyhow!("There was an error listing instances: {e}",));

    // 10. DescribeScalingActivities: list the scaling activities that have
    occurred for the group so far.
    // Bonus: use CloudWatch API to get and show some metrics collected for
    the group.
    // CW.ListMetrics with Namespace='AWS/AutoScaling' and
    Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': }]
    // CW.GetMetricStatistics with Statistics='Sum'. Start and End times must
    be in UTC!
    let activities = self

```



```

        .autoscaling
        .describe_scaling_activities()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .into_paginator()
        .items()
        .send()
        .collect::()
        .await
        .map_err(|e| {
            anyhow!(
                "There was an error retrieving scaling activities: {}",
                DisplayErrorContext(&e)
            )
        });
    };

    AutoScalingScenarioDescription {
        group,
        instances,
        activities,
    }
}

async fn get_group(&self) -> Result<AutoScalingGroup, ScenarioError> {
    let describe_auto_scaling_groups = self
        .autoscaling
        .describe_auto_scaling_groups()
        .auto_scaling_group_names(self.auto_scaling_group_name.clone())
        .send()
        .await;

    if let Err(err) = describe_auto_scaling_groups {
        return Err(ScenarioError::new(
            format!(
                "Failed to get status of autoscaling group {}",
                self.auto_scaling_group_name.clone()
            )
            .as_str(),
            &err,
        ));
    }

    let describe_auto_scaling_groups_output =
        describe_auto_scaling_groups.unwrap();
}

```

```

    let auto_scaling_groups =
describe_auto_scaling_groups_output.auto_scaling_groups();
    let auto_scaling_group = auto_scaling_groups.first();

    if auto_scaling_group.is_none() {
        return Err(ScenarioError::with(format!(
            "Could not find autoscaling group {}",
            self.auto_scaling_group_name.clone()
        )));
    }

    Ok(auto_scaling_group.unwrap().clone())
}

pub async fn wait_for_no_scaling(&self) -> Result<(), ScenarioError> {
    let waiter = Waiter::new();
    let mut scaling = true;
    while scaling {
        waiter.sleep().await?;
        let describe_activities = self
            .autoscaling
            .describe_scaling_activities()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .send()
            .await
            .map_err(|e| {
                ScenarioError::new("Failed to get autoscaling activities for
group", &e)
            })?;
        let activities = describe_activities.activities();
        trace!(
            "Waiting for no scaling found {} activities",
            activities.len()
        );
        scaling = activities.iter().any(|a| a.progress() < Some(100));
    }
    Ok(())
}

pub async fn wait_for_stable(&self, size: usize) -> Result<(), ScenarioError> {
    self.wait_for_no_scaling().await?;

    let mut group = self.get_group().await?;
    let mut count = count_group_instances(&group);

```

```

    let waiter = Waiter::new();
    while count != size {
        trace!("Waiting for stable {size} (current: {count})");
        waiter.sleep().await?;
        group = self.get_group().await?;
        count = count_group_instances(&group);
    }

    Ok(())
}

pub async fn list_instances(&self) -> Result<Vec<String>, ScenarioError> {
    // The direct way to list instances is by using DescribeAutoScalingGroup's
    instances property. However, this returns a Vec<Instance>, as opposed to a
    Vec<AutoScalingInstanceDetails>.
    // Ok(self.get_group().await?.instances.unwrap_or_default().map(|i|
    i.instance_id.clone().unwrap_or_default()).filter(|id| !id.is_empty()).collect())

    // Alternatively, and for the sake of example, DescribeAutoScalingInstances
    returns a list that can be filtered by the client.
    self.autoscaling
        .describe_auto_scaling_instances()
        .into_paginator()
        .items()
        .send()
        .try_collect()
        .await
        .map(|items| {
            items
                .into_iter()
                .filter(|i| {
                    i.auto_scaling_group_name.as_deref()
                        == Some(self.auto_scaling_group_name.as_str())
                })
                .map(|i| i.instance_id.unwrap_or_default())
                .filter(|id| !id.is_empty())
                .collect:::<Vec<String>>()
        })
        .map_err(|err| ScenarioError::new("Failed to get list of auto scaling
instances", &err))
}

pub async fn scale_min_size(&self, size: i32) -> Result<(), ScenarioError> {

```

```
    let update_group = self
        .autoscaling
        .update_auto_scaling_group()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .min_size(size)
        .send()
        .await;
    if let Err(err) = update_group {
        return Err(ScenarioError::new(
            format!("Failer to update group to min size ({size})").as_str(),
            &err,
        ));
    }
    Ok(())
}

pub async fn scale_max_size(&self, size: i32) -> Result<(), ScenarioError> {
    // 5. UpdateAutoScalingGroup: update max size to 3.
    let update_group = self
        .autoscaling
        .update_auto_scaling_group()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .max_size(size)
        .send()
        .await;
    if let Err(err) = update_group {
        return Err(ScenarioError::new(
            format!("Failed to update group to max size ({size})").as_str(),
            &err,
        ));
    }
    Ok(())
}

pub async fn scale_desired_capacity(&self, capacity: i32) -> Result<(),
ScenarioError> {
    // 7. SetDesiredCapacity: set desired capacity to 2.
    // Wait for a second instance to launch.
    let update_group = self
        .autoscaling
        .set_desired_capacity()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .desired_capacity(capacity)
        .send()
}
```

```

        .await;
    if let Err(err) = update_group {
        return Err(ScenarioError::new(
            format!("Failed to update group to desired capacity
({capacity}))").as_str(),
            &err,
        ));
    }
    Ok(())
}

pub async fn scale_group_to_zero(&self) -> Result<(), ScenarioError> {
    // If this fails it's fine, just means there are extra cloudwatch metrics
events for the scale-down.
    let _ = self
        .autoscaling
        .disable_metrics_collection()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .send()
        .await;

    // 12. DeleteAutoScalingGroup (to delete the group you must stop all
instances):
    // UpdateAutoScalingGroup with MinSize=0
    let update_group = self
        .autoscaling
        .update_auto_scaling_group()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .min_size(0)
        .desired_capacity(0)
        .send()
        .await;
    if let Err(err) = update_group {
        return Err(ScenarioError::new(
            "Failed to update group for scaling down&",
            &err,
        ));
    }
}

let stable = self.wait_for_stable(0).await;
if let Err(err) = stable {
    return Err(ScenarioError::with(format!(
        "Error while waiting for group to be stable on scale down: {err}"
    )));
}

```

```
    }

    Ok(())
}

pub async fn terminate_some_instance(&self) -> Result<(), ScenarioError> {
    // Retrieve a list of instances in the auto scaling group.
    let auto_scaling_group = self.get_group().await?;
    let instances = auto_scaling_group.instances();
    // Or use other logic to find an instance to terminate.
    let instance = instances.first();
    if let Some(instance) = instance {
        let instance_id = if let Some(instance_id) = instance.instance_id() {
            instance_id
        } else {
            return Err(ScenarioError::with("Missing instance id"));
        };
        let termination = self
            .ec2
            .terminate_instances()
            .instance_ids(instance_id)
            .send()
            .await;
        if let Err(err) = termination {
            Err(ScenarioError::new(
                "There was a problem terminating an instance",
                &err,
            ))
        } else {
            Ok(())
        }
    } else {
        Err(ScenarioError::with("There was no instance to terminate"))
    }
}

fn count_group_instances(group: &AutoScalingGroup) -> usize {
    group.instances.as_ref().map(|i| i.len()).unwrap_or(0)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하십시오.

- [CreateAutoScalingGroup](#)
- [DeleteAutoScalingGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAutoScalingInstances](#)
- [DescribeScalingActivities](#)
- [DisableMetricsCollection](#)
- [EnableMetricsCollection](#)
- [SetDesiredCapacity](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

## 작업

### CreateAutoScalingGroup

다음 코드 예시에서는 CreateAutoScalingGroup을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn create_group(client: &Client, name: &str, id: &str) -> Result<(), Error> {
    client
        .create_auto_scaling_group()
        .auto_scaling_group_name(name)
        .instance_id(id)
        .min_size(1)
        .max_size(5)
        .send()
        .await?;

    println!("Created AutoScaling group");
}
```

```
Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateAutoScalingGroup](#)을 참조하십시오.

## DeleteAutoScalingGroup

다음 코드 예시에서는 DeleteAutoScalingGroup을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn delete_group(client: &Client, name: &str, force: bool) -> Result<(), Error>
{
    client
        .delete_auto_scaling_group()
        .auto_scaling_group_name(name)
        .set_force_delete(if force { Some(true) } else { None })
        .send()
        .await?;

    println!("Deleted Auto Scaling group");

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteAutoScalingGroup](#)를 참조하십시오.

## DescribeAutoScalingGroups

다음 코드 예시에서는 DescribeAutoScalingGroups을 사용하는 방법을 보여 줍니다.



## SDK for Rust

**Note**

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

```

async fn list_groups(client: &Client) -> Result<(), Error> {
    let resp = client.describe_auto_scaling_groups().send().await?;

    println!("Groups:");

    let groups = resp.auto_scaling_groups();

    for group in groups {
        println!(
            "Name: {}",
            group.auto_scaling_group_name().unwrap_or("Unknown")
        );
        println!(
            "Arn: {}",
            group.auto_scaling_group_arn().unwrap_or("unknown"),
        );
        println!("Zones: {:?}", group.availability_zones(),);
        println!();
    }

    println!("Found {} group(s)", groups.len());

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeAutoScalingGroups](#)을 참조하십시오.

## DescribeAutoScalingInstances

다음 코드 예시에서는 DescribeAutoScalingInstances을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
pub async fn list_instances(&self) -> Result<Vec<String>, ScenarioError> {
    // The direct way to list instances is by using DescribeAutoScalingGroup's
    instances property. However, this returns a Vec<Instance>, as opposed to a
    Vec<AutoScalingInstanceDetails>.
    // Ok(self.get_group().await?.instances.unwrap_or_default().map(|i|
    i.instance_id.clone().unwrap_or_default()).filter(|id| !id.is_empty()).collect())

    // Alternatively, and for the sake of example, DescribeAutoScalingInstances
    returns a list that can be filtered by the client.
    self.autoscaling
        .describe_auto_scaling_instances()
        .into_paginator()
        .items()
        .send()
        .try_collect()
        .await
        .map(|items| {
            items
                .into_iter()
                .filter(|i| {
                    i.auto_scaling_group_name.as_deref()
                        == Some(self.auto_scaling_group_name.as_str())
                })
                .map(|i| i.instance_id.unwrap_or_default())
                .filter(|id| !id.is_empty())
                .collect:::<Vec<String>>()
        })
        .map_err(|err| ScenarioError::new("Failed to get list of auto scaling
instances", &err))
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeAutoScalingInstances](#)를 참조하십시오.

## DescribeScalingActivities

다음 코드 예시에서는 DescribeScalingActivities을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn describe_scenario(&self) -> AutoScalingScenarioDescription {
    let group = self
        .autoscaling
        .describe_auto_scaling_groups()
        .auto_scaling_group_names(self.auto_scaling_group_name.clone())
        .send()
        .await
        .map(|s| {
            s.auto_scaling_groups()
                .iter()
                .map(|s| {
                    format!(
                        "{}: {}",
                        s.auto_scaling_group_name().unwrap_or("Unknown"),
                        s.status().unwrap_or("Unknown")
                    )
                })
                .collect:::<Vec<String>>()
        })
        .map_err(|e| {
            ScenarioError::new("Failed to describe auto scaling groups for
scenario", &e)
        });

    let instances = self
        .list_instances()
        .await
        .map_err(|e| anyhow!("There was an error listing instances: {e}",));

    // 10. DescribeScalingActivities: list the scaling activities that have
    occurred for the group so far.
```

```

    // Bonus: use CloudWatch API to get and show some metrics collected for
    the group.
    // CW.ListMetrics with Namespace='AWS/AutoScaling' and
    Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': }]
    // CW.GetMetricStatistics with Statistics='Sum'. Start and End times must
    be in UTC!
    let activities = self
        .autoscaling
        .describe_scaling_activities()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .into_paginator()
        .items()
        .send()
        .collect::<Result<Vec<_>, _>>()
        .await
        .map_err(|e| {
            anyhow!(
                "There was an error retrieving scaling activities: {}",
                DisplayErrorContext(&e)
            )
        });

    AutoScalingScenarioDescription {
        group,
        instances,
        activities,
    }
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeScalingActivities](#)를 참조하십시오.

## DisableMetricsCollection

다음 코드 예시에서는 `DisableMetricsCollection`을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
// If this fails it's fine, just means there are extra cloudwatch metrics
events for the scale-down.
let _ = self
    .autoscaling
    .disable_metrics_collection()
    .auto_scaling_group_name(self.auto_scaling_group_name.clone())
    .send()
    .await;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DisableMetricsCollection](#)을 참조하십시오.

## EnableMetricsCollection

다음 코드 예시에서는 EnableMetricsCollection을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
let enable_metrics_collection = autoscaling
    .enable_metrics_collection()
    .auto_scaling_group_name(auto_scaling_group_name.as_str())
    .granularity("1Minute")
    .set_metrics(Some(vec![
        String::from("GroupMinSize"),
        String::from("GroupMaxSize"),
        String::from("GroupDesiredCapacity"),
        String::from("GroupInServiceInstances"),
        String::from("GroupTotalInstances"),
    ]))
    .send()
    .await;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [EnableMetricsCollection](#)을 참조하십시오.

## SetDesiredCapacity

다음 코드 예시에서는 SetDesiredCapacity을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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


```
pub async fn scale_desired_capacity(&self, capacity: i32) -> Result<(),
ScenarioError> {
    // 7. SetDesiredCapacity: set desired capacity to 2.
    // Wait for a second instance to launch.
    let update_group = self
        .autoscaling
        .set_desired_capacity()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .desired_capacity(capacity)
        .send()
        .await;
    if let Err(err) = update_group {
        return Err(ScenarioError::new(
            format!("Failed to update group to desired capacity
({capacity}))").as_str(),
            &err,
        ));
    }
    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [SetDesiredCapacity](#)를 참조하십시오.

## TerminateInstanceInAutoScalingGroup

다음 코드 예시에서는 TerminateInstanceInAutoScalingGroup을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
pub async fn terminate_some_instance(&self) -> Result<(), ScenarioError> {
    // Retrieve a list of instances in the auto scaling group.
    let auto_scaling_group = self.get_group().await?;
    let instances = auto_scaling_group.instances();
    // Or use other logic to find an instance to terminate.
    let instance = instances.first();
    if let Some(instance) = instance {
        let instance_id = if let Some(instance_id) = instance.instance_id() {
            instance_id
        } else {
            return Err(ScenarioError::with("Missing instance id"));
        };
        let termination = self
            .ec2
            .terminate_instances()
            .instance_ids(instance_id)
            .send()
            .await;
        if let Err(err) = termination {
            Err(ScenarioError::new(
                "There was a problem terminating an instance",
                &err,
            ))
        } else {
            Ok(())
        }
    } else {
        Err(ScenarioError::with("There was no instance to terminate"))
    }
}

pub async fn get_group(&self) -> Result<AutoScalingGroup, ScenarioError> {
    let describe_auto_scaling_groups = self
        .autoscaling
```

```

        .describe_auto_scaling_groups()
        .auto_scaling_group_names(self.auto_scaling_group_name.clone())
        .send()
        .await;

    if let Err(err) = describe_auto_scaling_groups {
        return Err(ScenarioError::new(
            format!(
                "Failed to get status of autoscaling group {}",
                self.auto_scaling_group_name.clone()
            )
            .as_str(),
            &err,
        ));
    }

    let describe_auto_scaling_groups_output =
describe_auto_scaling_groups.unwrap();
    let auto_scaling_groups =
describe_auto_scaling_groups_output.auto_scaling_groups();
    let auto_scaling_group = auto_scaling_groups.first();

    if auto_scaling_group.is_none() {
        return Err(ScenarioError::with(format!(
            "Could not find autoscaling group {}",
            self.auto_scaling_group_name.clone()
        )));
    }

    Ok(auto_scaling_group.unwrap().clone())
}

```


- API 세부 정보는 AWS SDK for Rust API 참조의 [TerminateInstanceInAutoScalingGroup](#)을 참조하십시오.

## UpdateAutoScalingGroup

다음 코드 예시에서는 UpdateAutoScalingGroup을 사용하는 방법을 보여 줍니다.



## SDK for Rust

 Note

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

```
async fn update_group(client: &Client, name: &str, size: i32) -> Result<(), Error> {
    client
        .update_auto_scaling_group()
        .auto_scaling_group_name(name)
        .max_size(size)
        .send()
        .await?;

    println!("Updated AutoScaling group");

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [UpdateAutoScalingGroup](#)을 참조하십시오.

## SDK for Rust를 사용한 Amazon Bedrock 런타임 예제

다음 코드 예제에서는 Amazon Bedrock 런타임에서 AWS SDK for Rust를 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

- [Anthropic Claude](#)

# Anthropic Claude

## Converse

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

## SDK for Rust

### Note

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

Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보냅니다.

```
#[tokio::main]
async fn main() -> Result<(), BedrockConverseError> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::defaults(BehaviorVersion::latest())
        .region(CLAUDE_REGION)
        .load()
        .await;
    let client = Client::new(&sdk_config);

    let response = client
        .converse()
        .model_id(MODEL_ID)
        .messages(
            Message::builder()
                .role(ConversationRole::User)
                .content(ContentBlock::Text(USER_MESSAGE.to_string()))
                .build()
                .map_err(|_| "failed to build message"?),
        )
        .send()
        .await;

    match response {
        Ok(output) => {
            let text = get_converse_output_text(output)?;
```

```

        println!("{}", text);
        Ok(())
    }
    Err(e) => Err(e
        .as_service_error()
        .map(BedrockConverseError::from)
        .unwrap_or_else(|| BedrockConverseError("Unknown service
error".into()))),
    }
}

fn get_converse_output_text(output: ConverseOutput) -> Result<String,
BedrockConverseError> {
    let text = output
        .output()
        .ok_or("no output")?
        .as_message()
        .map_err(|_| "output not a message")?
        .content()
        .first()
        .ok_or("no content in message")?
        .as_text()
        .map_err(|_| "content is not text")?
        .to_string();
    Ok(text)
}

```

문, 오류 유틸리티, 상수를 사용합니다.

```

use aws_config::BehaviorVersion;
use aws_sdk_bedrockruntime::{
    operation::converse::{ConverseError, ConverseOutput},
    types::{ContentBlock, ConversationRole, Message},
    Client,
};

// Set the model ID, e.g., Claude 3 Haiku.
const MODEL_ID: &str = "anthropic.claude-3-haiku-20240307-v1:0";
const CLAUDE_REGION: &str = "us-east-1";

// Start a conversation with the user message.

```

```

const USER_MESSAGE: &str = "Describe the purpose of a 'hello world' program in one
line.";

#[derive(Debug)]
struct BedrockConverseError(String);
impl std::fmt::Display for BedrockConverseError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "Can't invoke '{}'. Reason: {}", MODEL_ID, self.0)
    }
}
impl std::error::Error for BedrockConverseError {}
impl From<&str> for BedrockConverseError {
    fn from(value: &str) -> Self {
        BedrockConverseError(value.to_string())
    }
}
impl From<&ConverseError> for BedrockConverseError {
    fn from(value: &ConverseError) -> Self {
        BedrockConverseError::from(match value {
            ConverseError::ModelTimeoutException(_) => "Model took too long",
            ConverseError::ModelNotReadyException(_) => "Model is not ready",
            _ => "Unknown",
        })
    }
}
}

```

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

## ConverseStream

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

## SDK for Rust

### Note

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

Bedrock의 ConverseStream API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내고 응답 토큰을 스트리밍합니다.

```
#[tokio::main]
async fn main() -> Result<(), BedrockConverseStreamError> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::defaults(BehaviorVersion::latest())
        .region(CLAUDE_REGION)
        .load()
        .await;
    let client = Client::new(&sdk_config);

    let response = client
        .converse_stream()
        .model_id(MODEL_ID)
        .messages(
            Message::builder()
                .role(ConversationRole::User)
                .content(ContentBlock::Text(USER_MESSAGE.to_string()))
                .build()
                .map_err(|_| "failed to build message"?),
        )
        .send()
        .await;

    let mut stream = match response {
        Ok(output) => Ok(output.stream),
        Err(e) => Err(BedrockConverseStreamError::from(
            e.as_service_error().unwrap(),
        )),
    }?;

    loop {
        let token = stream.recv().await;
        match token {
            Ok(Some(text)) => {
                let next = get_converse_output_text(text)?;
                print!("{}", next);
                Ok(())
            }
            Ok(None) => break,
            Err(e) => Err(e
                .as_service_error())
        }
    }
}
```

```

        .map(BedrockConverseStreamError::from)
        .unwrap_or(BedrockConverseStreamError(
            "Unknown error receiving stream".into(),
        )),
    }?
}

println!();

Ok(())
}

fn get_converse_output_text(
    output: ConverseStreamOutputType,
) -> Result<String, BedrockConverseStreamError> {
    Ok(match output {
        ConverseStreamOutputType::ContentBlockDelta(event) => match event.delta() {
            Some(delta) => delta.as_text().cloned().unwrap_or_else(|_| "".into()),
            None => "".into(),
        },
        _ => "".into(),
    })
}

```

문, 오류 유틸리티, 상수를 사용합니다.

```

use aws_config::BehaviorVersion;
use aws_sdk_bedrockruntime::{
    error::ProvideErrorMetadata,
    operation::converse_stream::ConverseStreamError,
    types::{
        error::ConverseStreamOutputError, ContentBlock, ConversationRole,
        ConverseStreamOutput as ConverseStreamOutputType, Message,
    },
    Client,
};

// Set the model ID, e.g., Claude 3 Haiku.
const MODEL_ID: &str = "anthropic.claude-3-haiku-20240307-v1:0";
const CLAUDE_REGION: &str = "us-east-1";

```

```

// Start a conversation with the user message.
const USER_MESSAGE: &str = "Describe the purpose of a 'hello world' program in one
line.";

#[derive(Debug)]
struct BedrockConverseStreamError(String);
impl std::fmt::Display for BedrockConverseStreamError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "Can't invoke '{}'. Reason: {}", MODEL_ID, self.0)
    }
}
impl std::error::Error for BedrockConverseStreamError {}
impl From<&str> for BedrockConverseStreamError {
    fn from(value: &str) -> Self {
        BedrockConverseStreamError(value.into())
    }
}

impl From<&ConverseStreamError> for BedrockConverseStreamError {
    fn from(value: &ConverseStreamError) -> Self {
        BedrockConverseStreamError(
            match value {
                ConverseStreamError::ModelTimeoutException(_) => "Model took too
long",
                ConverseStreamError::ModelNotReadyException(_) => "Model is not
ready",
                _ => "Unknown",
            }
            .into(),
        )
    }
}

impl From<&ConverseStreamOutputError> for BedrockConverseStreamError {
    fn from(value: &ConverseStreamOutputError) -> Self {
        match value {
            ConverseStreamOutputError::ValidationException(ve) =>
                BedrockConverseStreamError(
                    ve.message().unwrap_or("Unknown ValidationException").into(),
                ),
            ConverseStreamOutputError::ThrottlingException(te) =>
                BedrockConverseStreamError(
                    te.message().unwrap_or("Unknown ThrottlingException").into(),
                ),
        }
    }
}

```

```

        value => BedrockConverseStreamError(
            value
                .message()
                .unwrap_or("Unknown StreamOutput exception")
                .into(),
        ),
    },
}
}
}

```

- API 세부 정보는 Rust API용 AWS SDK 참조의 [ConverseStream](#)를 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 Rust

#### Note

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

데모의 기본 시나리오 및 로직입니다. 사용자, Amazon Bedrock Converse API 및 날씨 도구 간의 대화를 오케스트레이션합니다.

```

#[derive(Debug)]
#[allow(dead_code)]
struct InvokeToolResult(String, ToolResultBlock);
struct ToolUseScenario {
    client: Client,
    conversation: Vec<Message>,
    system_prompt: SystemContentBlock,
    tool_config: ToolConfiguration,
}

impl ToolUseScenario {

```



```
fn new(client: Client) -> Self {
    let system_prompt = SystemContentBlock::Text(SYSTEM_PROMPT.into());
    let tool_config = ToolConfiguration::builder()
        .tools(Tool::ToolSpec(
            ToolSpecification::builder()
                .name(TOOL_NAME)
                .description(TOOL_DESCRIPTION)
                .input_schema(ToolInputSchema::Json(make_tool_schema()))
                .build()
                .unwrap(),
        ))
        .build()
        .unwrap();

    ToolUseScenario {
        client,
        conversation: vec![],
        system_prompt,
        tool_config,
    }
}

async fn run(&mut self) -> Result<(), ToolUseScenarioError> {
    loop {
        let input = get_input().await?;
        if input.is_none() {
            break;
        }

        let message = Message::builder()
            .role(User)
            .content(ContentBlock::Text(input.unwrap()))
            .build()
            .map_err(ToolUseScenarioError::from)?;
        self.conversation.push(message);

        let response = self.send_to_bedrock().await?;

        self.process_model_response(response).await?;
    }

    Ok(())
}
```

```
    async fn send_to_bedrock(&mut self) -> Result<ConverseOutput,
ToolUseScenarioError> {
        debug!("Sending conversation to bedrock");
        self.client
            .converse()
            .model_id(MODEL_ID)
            .set_messages(Some(self.conversation.clone()))
            .system(self.system_prompt.clone())
            .tool_config(self.tool_config.clone())
            .send()
            .await
            .map_err(ToolUseScenarioError::from)
    }

    async fn process_model_response(
        &mut self,
        mut response: ConverseOutput,
    ) -> Result<(), ToolUseScenarioError> {
        let mut iteration = 0;

        while iteration < MAX_RECURSIONS {
            iteration += 1;
            let message = if let Some(ref output) = response.output {
                if output.is_message() {
                    Ok(output.as_message().unwrap().clone())
                } else {
                    Err(ToolUseScenarioError(
                        "Converse Output is not a message".into(),
                    ))
                }
            } else {
                Err(ToolUseScenarioError("Missing Converse Output".into()))
            }?;

            self.conversation.push(message.clone());

            match response.stop_reason {
                StopReason::ToolUse => {
                    response = self.handle_tool_use(&message).await?;
                }
                StopReason::EndTurn => {
                    print_model_response(&message.content[0])?;
                    return Ok(());
                }
            }
        }
    }
}
```

```

        _ => (),
    }
}

Err(ToolUseScenarioError(
    "Exceeded MAX_ITERATIONS when calling tools".into(),
))
}

async fn handle_tool_use(
    &mut self,
    message: &Message,
) -> Result<ConverseOutput, ToolUseScenarioError> {
    let mut tool_results: Vec<ContentBlock> = vec![];

    for block in &message.content {
        match block {
            ContentBlock::Text(_) => print_model_response(block)?,
            ContentBlock::ToolUse(tool) => {
                let tool_response = self.invoke_tool(tool).await?;
                tool_results.push(ContentBlock::ToolResult(tool_response.1));
            }
            _ => (),
        };
    }

    let message = Message::builder()
        .role(User)
        .set_content(Some(tool_results))
        .build()?;
    self.conversation.push(message);

    self.send_to_bedrock().await
}

async fn invoke_tool(
    &mut self,
    tool: &ToolUseBlock,
) -> Result<InvokeToolResult, ToolUseScenarioError> {
    match tool.name() {
        TOOL_NAME => {
            println!(
                "\x1b[0;90mExecuting tool: {TOOL_NAME} with input: {:?}...
\x1b[0m",

```

```

        tool.input()
    );
    let content = fetch_weather_data(tool).await?;
    println!(
        "\x1b[0;90mTool responded with {:?}\x1b[0m",
        content.content()
    );
    Ok(InvokeToolResult(tool.tool_use_id.clone(), content))
}
_ => Err(ToolUseScenarioError(format!(
    "The requested tool with name {} does not exist",
    tool.name()
))),
)),
}
}
}

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::defaults(BehaviorVersion::latest())
        .region(CLAUDE_REGION)
        .load()
        .await;
    let client = Client::new(&sdk_config);

    let mut scenario = ToolUseScenario::new(client);

    header();
    if let Err(err) = scenario.run().await {
        println!("There was an error running the scenario! {}", err.0)
    }
    footer();
}

```

데모에서 사용하는 날씨 도구입니다. 이 스크립트는 도구 사양을 정의하고 Open-Meteo API를 사용하여 날씨 데이터를 검색하는 로직을 구현합니다.

```

const ENDPOINT: &str = "https://api.open-meteo.com/v1/forecast";
async fn fetch_weather_data(
    tool_use: &ToolUseBlock,
) -> Result<ToolResultBlock, ToolUseScenarioError> {

```

```
let input = tool_use.input();
let latitude = input
    .as_object()
    .unwrap()
    .get("latitude")
    .unwrap()
    .as_string()
    .unwrap();
let longitude = input
    .as_object()
    .unwrap()
    .get("longitude")
    .unwrap()
    .as_string()
    .unwrap();
let params = [
    ("latitude", latitude),
    ("longitude", longitude),
    ("current_weather", "true"),
];

debug!("Calling {ENDPOINT} with {params:?}");

let response = reqwest::Client::new()
    .get(ENDPOINT)
    .query(&params)
    .send()
    .await
    .map_err(|e| ToolUseScenarioError(format!("Error requesting weather:
{e:?}")))?
    .error_for_status()
    .map_err(|e| ToolUseScenarioError(format!("Failed to request weather:
{e:?}")))?;

debug!("Response: {response:?}");

let bytes = response
    .bytes()
    .await
    .map_err(|e| ToolUseScenarioError(format!("Error reading response:
{e:?}")))?;

let result = String::from_utf8(bytes.to_vec())
    .map_err(|_| ToolUseScenarioError("Response was not utf8".into()))?;
```

```
Ok(ToolResultBlock::builder()
    .tool_use_id(tool_use.tool_use_id())
    .content(ToolResultContentBlock::Text(result))
    .build()?)
}
```

메시지 콘텐츠 블록을 인쇄하는 유틸리티입니다.

```
fn print_model_response(block: &ContentBlock) -> Result<(), ToolUseScenarioError> {
    if block.is_text() {
        let text = block.as_text().unwrap();
        println!("\x1b[0;90mThe model's response:\x1b[0m\n{text}");
        Ok(())
    } else {
        Err(ToolUseScenarioError(format!(
            "Content block is not text ({block:?})"
        )))
    }
}
```

문, 오류 유틸리티, 상수를 사용합니다.

```
use std::{collections::HashMap, io::stdin};

use aws_config::BehaviorVersion;
use aws_sdk_bedrockruntime::{
    error::{BuildError, SdkError},
    operation::converse::{ConverseError, ConverseOutput},
    types::{
        ContentBlock, ConversationRole::User, Message, StopReason,
        SystemContentBlock, Tool,
        ToolConfiguration, ToolInputSchema, ToolResultBlock, ToolResultContentBlock,
        ToolSpecification, ToolUseBlock,
    },
    Client,
};
use aws_smithy_runtime_api::http::Response;
use aws_smithy_types::Document;
use tracing::debug;
```

```
// Set the model ID, e.g., Claude 3 Haiku.
const MODEL_ID: &str = "anthropic.claude-3-haiku-20240307-v1:0";
const CLAUDE_REGION: &str = "us-east-1";

const SYSTEM_PROMPT: &str = "You are a weather assistant that provides current
weather data for user-specified locations using only
the Weather_Tool, which expects latitude and longitude. Infer the coordinates from
the location yourself.
If the user provides coordinates, infer the approximate location and refer to it in
your response.
To use the tool, you strictly apply the provided tool specification.

- Explain your step-by-step process, and give brief updates before each step.
- Only use the Weather_Tool for data. Never guess or make up information.
- Repeat the tool use for subsequent requests if necessary.
- If the tool errors, apologize, explain weather is unavailable, and suggest other
options.
- Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports
concise. Sparingly use
emojis where appropriate.
- Only respond to weather queries. Remind off-topic users of your purpose.
- Never claim to search online, access external data, or use tools besides
Weather_Tool.
- Complete the entire process until you have all required data before sending the
complete response.
";

// The maximum number of recursive calls allowed in the tool_use_demo function.
// This helps prevent infinite loops and potential performance issues.
const MAX_RECURSIONS: i8 = 5;

const TOOL_NAME: &str = "Weather_Tool";
const TOOL_DESCRIPTION: &str =
    "Get the current weather for a given location, based on its WGS84 coordinates.";
fn make_tool_schema() -> Document {
    Document::Object(HashMap::<String, Document>::from([
        ("type".into(), Document::String("object".into())),
        (
            "properties".into(),
            Document::Object(HashMap::from([
                (
                    "latitude".into(),
                    Document::Object(HashMap::from([
                        ("type".into(), Document::String("string".into()))],
                    )
                )
            ]))
        )
    ]))
}
```

```

        (
            "description".into(),
            Document::String("Geographical WGS84 latitude of the
location.".into()),
        ),
    ])),
),
(
    "longitude".into(),
    Document::Object(HashMap::from([
        ("type".into(), Document::String("string".into())),
        (
            "description".into(),
            Document::String(
                "Geographical WGS84 longitude of the
location.".into()),
        ),
    ])),
),
]),
),
(
    "required".into(),
    Document::Array(vec![
        Document::String("latitude".into()),
        Document::String("longitude".into()),
    ]),
),
]))
}

#[derive(Debug)]
struct ToolUseScenarioError(String);
impl std::fmt::Display for ToolUseScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "Tool use error with '{}'. Reason: {}", MODEL_ID, self.0)
    }
}
impl From<&str> for ToolUseScenarioError {
    fn from(value: &str) -> Self {
        ToolUseScenarioError(value.into())
    }
}
}

```



```
impl From<BuildError> for ToolUseScenarioError {
    fn from(value: BuildError) -> Self {
        ToolUseScenarioError(value.to_string().clone())
    }
}
impl From<SdkError<ConverseError, Response>> for ToolUseScenarioError {
    fn from(value: SdkError<ConverseError, Response>) -> Self {
        ToolUseScenarioError(match value.as_service_error() {
            Some(value) => value.meta().message().unwrap_or("Unknown").into(),
            None => "Unknown".into(),
        })
    }
}
```

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

## SDK for Rust를 사용한 Amazon Cognito 자격 증명 공급자 예제

다음 코드 예제에서는 Amazon Cognito Identity Provider와 함께 AWS SDK for Rust를 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

#### ListUserPools

다음 코드 예시에서는 ListUserPools을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```

async fn show_pools(client: &Client) -> Result<(), Error> {
    let response = client.list_user_pools().max_results(10).send().await?;
    let pools = response.user_pools();
    println!("User pools:");
    for pool in pools {
        println!(" ID:           {}", pool.id().unwrap_or_default());
        println!(" Name:           {}", pool.name().unwrap_or_default());
        println!(" Lambda Config:  {:?}", pool.lambda_config().unwrap());
        println!(
            "   Last modified:  {}",
            pool.last_modified_date().unwrap().to_chrono_utc()?
        );
        println!(
            "   Creation date:  {:?}",
            pool.creation_date().unwrap().to_chrono_utc()
        );
        println!();
    }
    println!("Next token: {}", response.next_token().unwrap_or_default());

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListUserPools](#)를 참조하십시오.

## SDK for Rust를 사용한 Amazon Cognito Sync 예제

다음 코드 예제에서는 Amazon Cognito Sync와 함께 AWS SDK for Rust를 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

### ListIdentityPoolUsage

다음 코드 예시에서는 ListIdentityPoolUsage을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn show_pools(client: &Client) -> Result<(), Error> {
    let response = client
        .list_identity_pool_usage()
        .max_results(10)
        .send()
        .await?;

    let pools = response.identity_pool_usages();
    println!("Identity pools:");

    for pool in pools {
        println!(
            "  Identity pool ID:   {}",
            pool.identity_pool_id().unwrap_or_default()
        );
        println!(
            "  Data storage:       {}",

```

```

        pool.data_storage().unwrap_or_default()
    );
    println!(
        " Sync sessions count: {}",
        pool.sync_sessions_count().unwrap_or_default()
    );
    println!(
        " Last modified:          {}",
        pool.last_modified_date().unwrap().to_chrono_utc()?
    );
    println!();
}

println!("Next token: {}", response.next_token().unwrap_or_default());

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListIdentityPoolUsage](#)를 참조하십시오.

## SDK for Rust를 사용한 Firehose 예제

다음 코드 예제에서는 Firehose에서 AWS SDK for Rust를 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

#### PutRecordBatch

다음 코드 예시에서는 PutRecordBatch을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
async fn put_record_batch(
    client: &Client,
    stream: &str,
    data: Vec<Record>,
) -> Result<PutRecordBatchOutput, SdkError<PutRecordBatchError>> {
    client
        .put_record_batch()
        .delivery_stream_name(stream)
        .set_records(Some(data))
        .send()
        .await
}
```

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

## SDK for Rust를 사용한 Amazon DocumentDB 예제

다음 코드 예제에서는 Amazon DocumentDB와 함께 AWS SDK for Rust를 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 간접 호출

다음 코드 예제는 DocumentDB 변경 스트림에서 레코드를 수신하여 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DocumentDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

#### SDK for Rust

##### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

### Rust를 사용하여 Lambda로 Amazon DocumentDB 이벤트 소비

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
    ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(), Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
```

```
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())
}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}
```

## SDK for Rust를 사용한 DynamoDB 예제

다음 코드 예제에서는 DynamoDB에서 AWS SDK for Rust를 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.

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

AWS 커뮤니티 기여는 여러 팀이 생성하고 유지 관리하는 예입니다 AWS. 피드백을 제공하려면 연결된 리포지토리에 제공된 메커니즘을 사용합니다.

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

## 주제

- [작업](#)
- [시나리오](#)
- [서버리스 예제](#)
- [AWS 커뮤니티 기여](#)

## 작업

### CreateTable

다음 코드 예시에서는 CreateTable을 사용하는 방법을 보여 줍니다.

#### SDK for Rust

#### Note

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

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
```



```

let table_name: String = table.into();

let ad = AttributeDefinition::builder()
    .attribute_name(&a_name)
    .attribute_type(ScalarAttributeType::S)
    .build()
    .map_err(Error::BuildError)?;

let ks = KeySchemaElement::builder()
    .attribute_name(&a_name)
    .key_type(KeyType::Hash)
    .build()
    .map_err(Error::BuildError)?;

let pt = ProvisionedThroughput::builder()
    .read_capacity_units(10)
    .write_capacity_units(5)
    .build()
    .map_err(Error::BuildError)?;

let create_table_response = client
    .create_table()
    .table_name(table_name)
    .key_schema(ks)
    .attribute_definitions(ad)
    .provisioned_throughput(pt)
    .send()
    .await;

match create_table_response {
    Ok(out) => {
        println!("Added table {} with key {}", table, key);
        Ok(out)
    }
    Err(e) => {
        eprintln!("Got an error creating table:");
        eprintln!("{}", e);
        Err(Error::unhandled(e))
    }
}
}

```

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

## DeleteItem

다음 코드 예시에서는 DeleteItem을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn delete_item(
    client: &Client,
    table: &str,
    key: &str,
    value: &str,
) -> Result<DeleteItemOutput, Error> {
    match client
        .delete_item()
        .table_name(table)
        .key(key, AttributeValue::S(value.into()))
        .send()
        .await
    {
        Ok(out) => {
            println!("Deleted item from table");
            Ok(out)
        }
        Err(e) => Err(Error::unhandled(e)),
    }
}
```

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

## DeleteTable

다음 코드 예시에서는 DeleteTable을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn delete_table(client: &Client, table: &str) -> Result<DeleteTableOutput,
Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

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

**ListTables**

다음 코드 예시에서는 ListTables을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect:::<Result<Vec<_>, _>>().await?;
```

```
println!("Tables:");

for name in &table_names {
    println!("  {}", name);
}

println!("Found {} tables", table_names.len());
Ok(table_names)
}
```

테이블이 존재하는지 확인합니다.

```
pub async fn table_exists(client: &Client, table: &str) -> Result<bool, Error> {
    debug!("Checking for table: {table}");
    let table_list = client.list_tables().send().await;

    match table_list {
        Ok(list) => Ok(list.table_names().contains(&table.into())),
        Err(e) => Err(e.into()),
    }
}
```

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

## PutItem

다음 코드 예시에서는 PutItem을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
    Result<ItemOut, Error> {
```

```
let user_av = AttributeValue::S(item.username);
let type_av = AttributeValue::S(item.p_type);
let age_av = AttributeValue::S(item.age);
let first_av = AttributeValue::S(item.first);
let last_av = AttributeValue::S(item.last);

let request = client
    .put_item()
    .table_name(table)
    .item("username", user_av)
    .item("account_type", type_av)
    .item("age", age_av)
    .item("first_name", first_av)
    .item("last_name", last_av);

println!("Executing request [{request:?}] to add item...");

let resp = request.send().await?;

let attributes = resp.attributes().unwrap();

let username = attributes.get("username").cloned();
let first_name = attributes.get("first_name").cloned();
let last_name = attributes.get("last_name").cloned();
let age = attributes.get("age").cloned();
let p_type = attributes.get("p_type").cloned();

println!(
    "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
    username, first_name, last_name, age, p_type
);

Ok(ItemOut {
    p_type,
    age,
    username,
    first_name,
    last_name,
})
}
```

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

## Query

다음 코드 예시에서는 Query를 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

지정된 연도에 제작된 영화를 찾습니다.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy", AttributeValue::N(year.to_string()))
        .send()
        .await?;


    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

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

## Scan

다음 코드 예시에서는 Scan을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
pub async fn list_items(client: &Client, table: &str, page_size: Option<i32>) ->
Result<(), Error> {
    let page_size = page_size.unwrap_or(10);
    let items: Result<Vec<_>, _> = client
        .scan()
        .table_name(table)
        .limit(page_size)
        .into_paginator()
        .items()
        .send()
        .collect()
        .await;

    println!("Items in table (up to {page_size}):");
    for item in items? {
        println!("  {:?}", item);
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [Scan](#)을 참조하십시오.


## 시나리오

### 로컬 인스턴스에 연결

다음 코드 예제는 엔드포인트 URL을 재정의하여 DynamoDB 및 AWS SDK의 로컬 개발 배포에 연결하는 방법을 보여줍니다.

자세한 내용은 [DynamoDB Local](#)을 참조하세요.

## SDK for Rust

 Note

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

```
/// Lists your tables from a local DynamoDB instance by setting the SDK Config's
/// endpoint_url and test_credentials.
#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();

    let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
        .test_credentials()
        // DynamoDB run locally uses port 8000 by default.
        .endpoint_url("http://localhost:8000")
        .load()
        .await;
    let dynamodb_local_config =
aws_sdk_dynamodb::config::Builder::from(&config).build();

    let client = aws_sdk_dynamodb::Client::from_conf(dynamodb_local_config);

    let list_resp = client.list_tables().send().await;
    match list_resp {
        Ok(resp) => {
            println!("Found {} tables", resp.table_names().len());
            for name in resp.table_names() {
                println!("  {}", name);
            }
        }
        Err(err) => eprintln!("Failed to list local dynamodb tables: {err:?}"),
    }
}
```



## 사진을 관리하기 위한 서버리스 애플리케이션 만들기

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

### SDK for Rust

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 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

### 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 Rust

#### Note

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

```

async fn make_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<(), SdkError<CreateTableError>> {
    let ad = AttributeDefinition::builder()
        .attribute_name(key)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .expect("creating AttributeDefinition");

    let ks = KeySchemaElement::builder()
        .attribute_name(key)
        .key_type(KeyType::Hash)
        .build()
        .expect("creating KeySchemaElement");

    let pt = ProvisionedThroughput::builder()
        .read_capacity_units(10)
        .write_capacity_units(5)
        .build()
        .expect("creating ProvisionedThroughput");

    match client
        .create_table()
        .table_name(table)
        .key_schema(ks)
        .attribute_definitions(ad)
        .provisioned_throughput(pt)
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn add_item(client: &Client, item: Item) -> Result<(),
SdkError<ExecuteStatementError>> {
    match client
        .execute_statement()
        .statement(format!(
            r#"INSERT INTO "{}" VALUE {{

```

```

        "{}": ?,
        "account_type": ?,
        "age": ?,
        "first_name": ?,
        "last_name": ?
    }} "#,
        item.table, item.key
    ))
    .set_parameters(Some(vec![
        AttributeValue::S(item.utype),
        AttributeValue::S(item.age),
        AttributeValue::S(item.first_name),
        AttributeValue::S(item.last_name),
    ]))
    .send()
    .await
{
    Ok(_) => Ok(()),
    Err(e) => Err(e),
}
}

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {

```

```

        println!("Got an error querying table:");
        println!("{}", e);
        process::exit(1);
    }
}

}

async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ExecuteStatement](#)을 참조하십시오.

## EXIF 및 기타 이미지 정보 저장

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져옵니다.
- Amazon S3 버킷에 이미지 파일을 업로드합니다.
- Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(레이블)을 파악합니다.
- EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

## SDK for Rust

JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져오고, 이미지 파일을 Amazon S3 버킷에 업로드 하며, Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(Amazon Rekognition의 레이블)을 파악한 후 EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon Rekognition
- Amazon S3

## 서버리스 예제

DynamoDB 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제는 DynamoDB 스트림에서 레코드를 수신하여 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DynamoDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

## SDK for Rust

### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
```

```

//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
  ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())
}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);

```

```

    lambda_runtime::run(func).await?;
    Ok(())
}

```

## DynamoDB 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제는 DynamoDB 스트림에서 이벤트를 수신하는 Lambda 함수에 대해 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

## SDK for Rust

### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```

use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
    Result<DynamoDbEventResponse, Error> {

```

```

let mut response = DynamoDbEventResponse {
    batch_item_failures: vec![],
};

let records = &event.payload.records;

if records.is_empty() {
    tracing::info!("No records found. Exiting.");
    return Ok(response);
}

for record in records {
    tracing::info!("EventId: {}", record.event_id);

    // Couldn't find a sequence number
    if record.change.sequence_number.is_none() {
        response.batch_item_failures.push(DynamoDbBatchItemFailure {
            item_identifier: Some("").to_string(),
        });
        return Ok(response);
    }

    // Process your record here...
    if process_record(record).is_err() {
        response.batch_item_failures.push(DynamoDbBatchItemFailure {
            item_identifier: record.change.sequence_number.clone(),
        });
        /* 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 Ok(response);
    }
}

tracing::info!("Successfully processed {} record(s)", records.len());

Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)

```



```
// disable printing the name of the module in every log line.
.with_target(false)
// disabling time is handy because CloudWatch will add the ingestion time.
.without_time()
.init();

run(service_fn(function_handler)).await
}
```

## AWS 커뮤니티 기여

### 서버리스 애플리케이션 빌드 및 테스트

다음 코드 예제는 Lambda 및 DynamoDB와 함께 API Gateway를 사용하여 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

#### SDK for Rust

Rust SDK를 사용하여 Lambda 및 DynamoDB가 포함된 API 게이트웨이로 구성된 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

## SDK for Rust를 사용한 Amazon EBS 예제

다음 코드 예제에서는 Amazon EBS에서 AWS SDK for Rust를 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

### CompleteSnapshot

다음 코드 예시에서는 CompleteSnapshot을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn finish(client: &Client, id: &str) -> Result<(), Error> {
    client
        .complete_snapshot()
        .changed_blocks_count(2)
        .snapshot_id(id)
        .send()
        .await?;

    println!("Snapshot ID {}", id);
    println!("The state is 'completed' when all of the modified blocks have been
transferred to Amazon S3.");
    println!("Use the get-snapshot-state code example to get the state of the
snapshot.");

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CompleteSnapshot](#)을 참조하십시오.

## PutSnapshotBlock

다음 코드 예시에서는 PutSnapshotBlock을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn add_block(
    client: &Client,
    id: &str,
    idx: usize,
    block: Vec<u8>,
    checksum: &str,
) -> Result<(), Error> {
    client
        .put_snapshot_block()
        .snapshot_id(id)
        .block_index(idx as i32)
        .block_data(ByteStream::from(block))
        .checksum(checksum)
        .checksum_algorithm(ChecksumAlgorithm::ChecksumAlgorithmSha256)
        .data_length(EBS_BLOCK_SIZE as i32)
        .send()
        .await?;

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [PutSnapshotBlock](#)을 참조하십시오.

## StartSnapshot

다음 코드 예시에서는 StartSnapshot을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn start(client: &Client, description: &str) -> Result<String, Error> {
    let snapshot = client
        .start_snapshot()
        .description(description)
        .encrypted(false)
        .volume_size(1)
        .send()
        .await?;

    Ok(snapshot.snapshot_id.unwrap())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [StartSnapshot](#)을 참조하십시오.

## SDK for Rust를 사용한 Amazon EC2 예제

다음 코드 예제에서는 Amazon EC2에서 AWS SDK for Rust를 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.

시작

## Hello Amazon EC2

다음 코드 예제는 Amazon EC2 사용을 시작하는 방법을 보여줍니다.

### SDK for Rust

#### Note

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

```
async fn show_security_groups(client: &aws_sdk_ec2::Client, group_ids: Vec<String>)
{
    let response = client
        .describe_security_groups()
        .set_group_ids(Some(group_ids))
        .send()
        .await;

    match response {
        Ok(output) => {
            for group in output.security_groups() {
                println!(
                    "Found Security Group {} ({}), vpc id {} and description {}",
                    group.group_name().unwrap_or("unknown"),
                    group.group_id().unwrap_or("id-unknown"),
                    group.vpc_id().unwrap_or("vpcid-unknown"),
                    group.description().unwrap_or("(none)")
                );
            }
        }
        Err(err) => {
            let err = err.into_service_error();
            let meta = err.meta();
            let message = meta.message().unwrap_or("unknown");
            let code = meta.code().unwrap_or("unknown");
            eprintln!("Error listing EC2 Security Groups: ({}code) {}message");
        }
    }
}
```

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

## 주제

- [기본 사항](#)
- [작업](#)

## 기본 사항

### 기본 사항 알아보기

다음 코드 예제는 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 Rust

#### Note

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

EC2InstanceScenario 구현에는 예제를 전체적으로 실행하는 로직이 포함되어 있습니다.

```

//! Scenario that uses the AWS SDK for Rust (the SDK) with Amazon Elastic Compute
  Cloud
//! (Amazon EC2) to do the following:
//!
//! * Create a key pair that is used to secure SSH communication between your
  computer and
//!   an EC2 instance.
//! * Create a security group that acts as a virtual firewall for your EC2 instances
  to

```

```
#!/ control incoming and outgoing traffic.
#!/ * Find an Amazon Machine Image (AMI) and a compatible instance type.
#!/ * Create an instance that is created from the instance type and AMI you select,
    and
#!/ is configured to use the security group and key pair created in this example.
#!/ * Stop and restart the instance.
#!/ * Create an Elastic IP address and associate it as a consistent IP address for
    your instance.
#!/ * Connect to your instance with SSH, using both its public IP address and your
    Elastic IP
    address.
#!/ * Clean up all of the resources created by this example.

use std::net::Ipv4Addr;

use crate::{
    ec2::{EC2Error, EC2},
    getting_started::{key_pair::KeyPairManager, util::Util},
    ssm::SSM,
};
use aws_sdk_ssm::types::Parameter;

use super::{
    elastic_ip::ElasticIpManager, instance::InstanceManager,
    security_group::SecurityGroupManager,
    util::ScenarioImage,
};

pub struct Ec2InstanceScenario {
    ec2: EC2,
    ssm: SSM,
    util: Util,
    key_pair_manager: KeyPairManager,
    security_group_manager: SecurityGroupManager,
    instance_manager: InstanceManager,
    elastic_ip_manager: ElasticIpManager,
}

impl Ec2InstanceScenario {
    pub fn new(ec2: EC2, ssm: SSM, util: Util) -> Self {
        Ec2InstanceScenario {
            ec2,
            ssm,
            util,
        }
    }
}
```

```

        key_pair_manager: Default::default(),
        security_group_manager: Default::default(),
        instance_manager: Default::default(),
        elastic_ip_manager: Default::default(),
    }
}

pub async fn run(&mut self) -> Result<(), EC2Error> {
    self.create_and_list_key_pairs().await?;
    self.create_security_group().await?;
    self.create_instance().await?;
    self.stop_and_start_instance().await?;
    self.associate_elastic_ip().await?;
    self.stop_and_start_instance().await?;
    Ok(())
}

/// 1. Creates an RSA key pair and saves its private key data as a .pem file in
secure
/// temporary storage. The private key data is deleted after the example
completes.
/// 2. Optionally, lists the first five key pairs for the current account.
pub async fn create_and_list_key_pairs(&mut self) -> Result<(), EC2Error> {
    println!( "Let's create an RSA key pair that you can be use to securely
connect to your EC2 instance.");

    let key_name = self.util.prompt_key_name()?;

    self.key_pair_manager
        .create(&self.ec2, &self.util, key_name)
        .await?;

    println!(
        "Created a key pair {} and saved the private key to {:?}.",
        self.key_pair_manager
            .key_pair()
            .key_name()
            .ok_or_else(|| EC2Error::new("No key name after creating key"))?,
        self.key_pair_manager
            .key_file_path()
            .ok_or_else(|| EC2Error::new("No key file after creating key"))?
    );

    if self.util.should_list_key_pairs()? {

```



```

        for pair in self.key_pair_manager.list(&self.ec2).await? {
            println!(
                "Found {:?} key {} with fingerprint:\t{:?}",
                pair.key_type(),
                pair.key_name().unwrap_or("Unknown"),
                pair.key_fingerprint()
            );
        }
    }

    Ok(())
}

/// 1. Creates a security group for the default VPC.
/// 2. Adds an inbound rule to allow SSH. The SSH rule allows only
///     inbound traffic from the current computer's public IPv4 address.
/// 3. Displays information about the security group.
///
/// This function uses <http://checkip.amazonaws.com> to get the current public
IP
/// address of the computer that is running the example. This method works in
most
/// cases. However, depending on how your computer connects to the internet, you
/// might have to manually add your public IP address to the security group by
using
/// the AWS Management Console.
pub async fn create_security_group(&mut self) -> Result<(), EC2Error> {
    println!("Let's create a security group to manage access to your
instance.");
    let group_name = self.util.prompt_security_group_name()?;

    self.security_group_manager
        .create(
            &self.ec2,
            &group_name,
            "Security group for example: get started with instances.",
        )
        .await?;

    println!(
        "Created security group {} in your default VPC {}.",
        self.security_group_manager.group_name(),
        self.security_group_manager
            .vpc_id()
    )
}

```

```

        .unwrap_or("(unknown vpc)")
    );

    let check_ip = self.util.do_get("https://checkip.amazonaws.com").await?;
    let current_ip_address: Ipv4Addr = check_ip.trim().parse().map_err(|e| {
        EC2Error::new(format!(
            "Failed to convert response {} to IP Address: {e:?}",
            check_ip
        ))
    })?;

    println!("Your public IP address seems to be {current_ip_address}");
    if self.util.should_add_to_security_group() {
        match self
            .security_group_manager
            .authorize_ingress(&self.ec2, current_ip_address)
            .await
        {
            Ok(_) => println!("Security group rules updated"),
            Err(err) => eprintln!("Couldn't update security group rules:
{err:?}"),
        }
    }
    println!("{}", self.security_group_manager);

    Ok(())
}

/// 1. Gets a list of Amazon Linux 2 AMIs from AWS Systems Manager. Specifying
the
///     '/aws/service/ami-amazon-linux-latest' path returns only the latest AMIs.
/// 2. Gets and displays information about the available AMIs and lets you
select one.
/// 3. Gets a list of instance types that are compatible with the selected AMI
and
///     lets you select one.
/// 4. Creates an instance with the previously created key pair and security
group,
///     and the selected AMI and instance type.
/// 5. Waits for the instance to be running and then displays its information.
pub async fn create_instance(&mut self) -> Result<(), EC2Error> {
    let ami = self.find_image().await?;

    let instance_types = self

```

```

        .ec2
        .list_instance_types(&ami.0)
        .await
        .map_err(|e| e.add_message("Could not find instance types"))?;
println!(
    "There are several instance types that support the {} architecture of
the image.",
    ami.0
        .architecture
        .as_ref()
        .ok_or_else(|| EC2Error::new(format!("Missing architecture in {:?}",
ami.0)))?
    );
let instance_type = self.util.select_instance_type(instance_types)?;

println!("Creating your instance and waiting for it to start...");
self.instance_manager
    .create(
        &self.ec2,
        ami.0
            .image_id()
            .ok_or_else(|| EC2Error::new("Could not find image ID"))?,
        instance_type,
        self.key_pair_manager.key_pair(),
        self.security_group_manager
            .security_group()
            .map(|sg| vec![sg])
            .ok_or_else(|| EC2Error::new("Could not find security group"))?,
    )
    .await
    .map_err(|e| e.add_message("Scenario failed to create instance"))?;

while let Err(err) = self
    .ec2
    .wait_for_instance_ready(self.instance_manager.instance_id(), None)
    .await
    {
        println!("{err}");
        if !self.util.should_continue_waiting() {
            return Err(err);
        }
    }

println!("Your instance is ready:\n{}", self.instance_manager);

```

```

        self.display_ssh_info();

        Ok(())
    }

    async fn find_image(&mut self) -> Result<ScenarioImage, EC2Error> {
        let params: Vec<Parameter> = self
            .ssm
            .list_path("/aws/service/ami-amazon-linux-latest")
            .await
            .map_err(|e| e.add_message("Could not find parameters for available
images"))?
            .into_iter()
            .filter(|param| param.name().is_some_and(|name| name.contains("amzn2")))
            .collect();
        let amzn2_images: Vec<ScenarioImage> = self
            .ec2
            .list_images(params)
            .await
            .map_err(|e| e.add_message("Could not find images"))?
            .into_iter()
            .map(ScenarioImage::from)
            .collect();
        println!("We will now create an instance from an Amazon Linux 2 AMI");
        let ami = self.util.select_scenario_image(amzn2_images)?;
        Ok(ami)
    }

    // 1. Stops the instance and waits for it to stop.
    // 2. Starts the instance and waits for it to start.
    // 3. Displays information about the instance.
    // 4. Displays an SSH connection string. When an Elastic IP address is
associated
//     with the instance, the IP address stays consistent when the instance stops
//     and starts.
    pub async fn stop_and_start_instance(&self) -> Result<(), EC2Error> {
        println!("Let's stop and start your instance to see what changes.");
        println!("Stopping your instance and waiting until it's stopped...");
        self.instance_manager.stop(&self.ec2).await?;
        println!("Your instance is stopped. Restarting...");
        self.instance_manager.start(&self.ec2).await?;
        println!("Your instance is running.");
        println!("{}", self.instance_manager);
    }

```

```

        if self.elastic_ip_manager.public_ip() == "0.0.0.0" {
            println!("Every time your instance is restarted, its public IP address
changes.");
        } else {
            println!(
                "Because you have associated an Elastic IP with your instance, you
can connect by using a consistent IP address after the instance restarts."
            );
        }
        self.display_ssh_info();
        Ok(())
    }

    /// 1. Allocates an Elastic IP address and associates it with the instance.
    /// 2. Displays an SSH connection string that uses the Elastic IP address.
    async fn associate_elastic_ip(&mut self) -> Result<(), EC2Error> {
        self.elastic_ip_manager.allocate(&self.ec2).await?;
        println!(
            "Allocated static Elastic IP address: {}",
            self.elastic_ip_manager.public_ip()
        );

        self.elastic_ip_manager
            .associate(&self.ec2, self.instance_manager.instance_id())
            .await?;
        println!("Associated your Elastic IP with your instance.");
        println!("You can now use SSH to connect to your instance by using the
Elastic IP.");
        self.display_ssh_info();
        Ok(())
    }

    /// Displays an SSH connection string that can be used to connect to a running
    /// instance.
    fn display_ssh_info(&self) {
        let ip_addr = if self.elastic_ip_manager.has_allocation() {
            self.elastic_ip_manager.public_ip()
        } else {
            self.instance_manager.instance_ip()
        };
        let key_file_path = self.key_pair_manager.key_file_path().unwrap();
        println!("To connect, open another command prompt and run the following
command:");
        println!("\nssh -i {} ec2-user@{ip_addr}\n", key_file_path.display());
    }

```

```
        let _ = self.util.enter_to_continue();
    }

    /// 1. Disassociate and delete the previously created Elastic IP.
    /// 2. Terminate the previously created instance.
    /// 3. Delete the previously created security group.
    /// 4. Delete the previously created key pair.
    pub async fn clean_up(self) {
        println!("Let's clean everything up. This example created these
resources:");
        println!(
            "\tKey pair: {}",
            self.key_pair_manager
                .key_pair()
                .key_name()
                .unwrap_or("(unknown key pair)")
        );
        println!(
            "\tSecurity group: {}",
            self.security_group_manager.group_name()
        );
        println!(
            "\tInstance: {}",
            self.instance_manager.instance_display_name()
        );
        if self.util.should_clean_resources() {
            if let Err(err) = self.elastic_ip_manager.remove(&self.ec2).await {
                eprintln!("{err}")
            }
            if let Err(err) = self.instance_manager.delete(&self.ec2).await {
                eprintln!("{err}")
            }
            if let Err(err) = self.security_group_manager.delete(&self.ec2).await {
                eprintln!("{err}");
            }
            if let Err(err) = self.key_pair_manager.delete(&self.ec2,
&self.util).await {
                eprintln!("{err}");
            }
        } else {
            println!("Ok, not cleaning up any resources!");
        }
    }
}
```

```

pub async fn run(mut scenario: Ec2InstanceScenario) {
    println!
    ("-----");
    println!(
        "Welcome to the Amazon Elastic Compute Cloud (Amazon EC2) get started with
instances demo."
    );
    println!
    ("-----");

    if let Err(err) = scenario.run().await {
        eprintln!("There was an error running the scenario: {err}")
    }

    println!
    ("-----");

    scenario.clean_up().await;

    println!("Thanks for running!");
    println!
    ("-----");
}

```

EC2Impl 구조체는 테스트를 위한 오토모크 지점 역할을 하며 해당 함수는 EC2 SDK 호출을 래핑합니다.

```

use std::{net::Ipv4Addr, time::Duration};

use aws_sdk_ec2::{
    client::Waiters,
    error::ProvideErrorMetadata,
    operation::{
        allocate_address::AllocateAddressOutput,
        associate_address::AssociateAddressOutput,
    },
    types::{
        DomainType, Filter, Image, Instance, InstanceType, IpPermission, IpRange,
        KeyPairInfo,
        SecurityGroup, Tag,
    },
};

```

```

    },
    Client as EC2Client,
};
use aws_sdk_ssm::types::Parameter;
use aws_smithy_runtime_api::client::waiters::error::WaiterError;

#[cfg(test)]
use mockall::automock;

#[cfg(not(test))]
pub use EC2Impl as EC2;

#[cfg(test)]
pub use MockEC2Impl as EC2;

#[derive(Clone)]
pub struct EC2Impl {
    pub client: EC2Client,
}

#[cfg_attr(test, automock)]
impl EC2Impl {
    pub fn new(client: EC2Client) -> Self {
        EC2Impl { client }
    }

    pub async fn create_key_pair(&self, name: String) -> Result<(KeyPairInfo,
String), EC2Error> {
        tracing::info!("Creating key pair {name}");
        let output = self.client.create_key_pair().key_name(name).send().await?;
        let info = KeyPairInfo::builder()
            .set_key_name(output.key_name)
            .set_key_fingerprint(output.key_fingerprint)
            .set_key_pair_id(output.key_pair_id)
            .build();
        let material = output
            .key_material
            .ok_or_else(|| EC2Error::new("Create Key Pair has no key material"))?;
        Ok((info, material))
    }

    pub async fn list_key_pair(&self) -> Result<Vec<KeyPairInfo>, EC2Error> {
        let output = self.client.describe_key_pairs().send().await?;
        Ok(output.key_pairs.unwrap_or_default())
    }
}

```



```
}

pub async fn delete_key_pair(&self, key_name: &str) -> Result<(), EC2Error> {
    let key_name: String = key_name.into();
    tracing::info!("Deleting key pair {key_name}");
    self.client
        .delete_key_pair()
        .key_name(key_name)
        .send()
        .await?;
    Ok(())
}

pub async fn create_security_group(
    &self,
    name: &str,
    description: &str,
) -> Result<SecurityGroup, EC2Error> {
    tracing::info!("Creating security group {name}");
    let create_output = self
        .client
        .create_security_group()
        .group_name(name)
        .description(description)
        .send()
        .await
        .map_err(EC2Error::from)?;

    let group_id = create_output
        .group_id
        .ok_or_else(|| EC2Error::new("Missing security group id after
creation"))?;

    let group = self
        .describe_security_group(&group_id)
        .await?
        .ok_or_else(|| {
            EC2Error::new(format!("Could not find security group with id
{group_id}"))
        })?;

    tracing::info!("Created security group {name} as {group_id}");

    Ok(group)
}
```

```

}

/// Find a single security group, by ID. Returns Err if multiple groups are
found.
pub async fn describe_security_group(
    &self,
    group_id: &str,
) -> Result<Option<SecurityGroup>, EC2Error> {
    let group_id: String = group_id.into();
    let describe_output = self
        .client
        .describe_security_groups()
        .group_ids(&group_id)
        .send()
        .await?;

    let mut groups = describe_output.security_groups.unwrap_or_default();

    match groups.len() {
        0 => Ok(None),
        1 => Ok(Some(groups.remove(0))),
        _ => Err(EC2Error::new(format!(
            "Expected single group for {group_id}"
        ))),
    }
}

/// Add an ingress rule to a security group explicitly allowing IPv4 address
/// as {ip}/32 over TCP port 22.
pub async fn authorize_security_group_ssh_ingress(
    &self,
    group_id: &str,
    ingress_ips: Vec<Ipv4Addr>,
) -> Result<(), EC2Error> {
    tracing::info!("Authorizing ingress for security group {group_id}");
    self.client
        .authorize_security_group_ingress()
        .group_id(group_id)
        .set_ip_permissions(Some(
            ingress_ips
                .into_iter()
                .map(|ip| {
                    IpPermission::builder()
                        .ip_protocol("tcp")

```

```

        .from_port(22)
        .to_port(22)
        .ip_ranges(IpRange::builder().cidr_ip(format!
({ip}/32)).build())
        .build()
    ))
    .collect(),
))
.send()
.await?;
Ok(())
}

pub async fn delete_security_group(&self, group_id: &str) -> Result<(),
EC2Error> {
    tracing::info!("Deleting security group {group_id}");
    self.client
        .delete_security_group()
        .group_id(group_id)
        .send()
        .await?;
    Ok(())
}

pub async fn list_images(&self, ids: Vec<Parameter>) -> Result<Vec<Image>,
EC2Error> {
    let image_ids = ids.into_iter().filter_map(|p| p.value).collect();
    let output = self
        .client
        .describe_images()
        .set_image_ids(Some(image_ids))
        .send()
        .await?;

    let images = output.images.unwrap_or_default();
    if images.is_empty() {
        Err(EC2Error::new("No images for selected AMIs"))
    } else {
        Ok(images)
    }
}

/// List instance types that match an image's architecture and are free tier
eligible.

```

```

pub async fn list_instance_types(&self, image: &Image) ->
Result<Vec<InstanceType>, EC2Error> {
    let architecture = format!(
        "{}",
        image.architecture().ok_or_else(|| EC2Error::new(format!(
            "Image {:?} does not have a listed architecture",
            image.image_id()
        )))?
    );
    let free_tier_eligible_filter = Filter::builder()
        .name("free-tier-eligible")
        .values("false")
        .build();
    let supported_architecture_filter = Filter::builder()
        .name("processor-info.supported-architecture")
        .values(architecture)
        .build();
    let response = self
        .client
        .describe_instance_types()
        .filters(free_tier_eligible_filter)
        .filters(supported_architecture_filter)
        .send()
        .await?;

    Ok(response
        .instance_types
        .unwrap_or_default()
        .into_iter()
        .filter_map(|iti| iti.instance_type)
        .collect())
}

pub async fn create_instance<'a>(
    &self,
    image_id: &'a str,
    instance_type: InstanceType,
    key_pair: &'a KeyPairInfo,
    security_groups: Vec<&'a SecurityGroup>,
) -> Result<String, EC2Error> {
    let run_instances = self
        .client
        .run_instances()
        .image_id(image_id)

```

```
        .instance_type(instance_type)
        .key_name(
            key_pair
                .key_name()
                .ok_or_else(|| EC2Error::new("Missing key name when launching
instance"))?),
        )
        .set_security_group_ids(Some(
            security_groups
                .iter()
                .filter_map(|sg| sg.group_id.clone())
                .collect(),
        ))
        .min_count(1)
        .max_count(1)
        .send()
        .await?;

if run_instances.instances().is_empty() {
    return Err(EC2Error::new("Failed to create instance"));
}

let instance_id = run_instances.instances()[0].instance_id().unwrap();
let response = self
    .client
    .create_tags()
    .resources(instance_id)
    .tags(
        Tag::builder()
            .key("Name")
            .value("From SDK Examples")
            .build(),
    )
    .send()
    .await;

match response {
    Ok(_) => tracing::info!("Created {instance_id} and applied tags."),
    Err(err) => {
        tracing::info!("Error applying tags to {instance_id}: {err:?}");
        return Err(err.into());
    }
}
```

```

        tracing::info!("Instance is created.");

        Ok(instance_id.to_string())
    }

    /// Wait for an instance to be ready and status ok (default wait 60 seconds)
    pub async fn wait_for_instance_ready(
        &self,
        instance_id: &str,
        duration: Option<Duration>,
    ) -> Result<(), EC2Error> {
        self.client
            .wait_until_instance_status_ok()
            .instance_ids(instance_id)
            .wait(duration.unwrap_or(Duration::from_secs(60)))
            .await
            .map_err(|err| match err {
                WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                    "Exceeded max time ({}s) waiting for instance to start.",
                    exceeded.max_wait().as_secs()
                )),
                _ => EC2Error::from(err),
            })?;
        Ok(())
    }

    pub async fn describe_instance(&self, instance_id: &str) -> Result<Instance,
    EC2Error> {
        let response = self
            .client
            .describe_instances()
            .instance_ids(instance_id)
            .send()
            .await?;

        let instance = response
            .reservations()
            .first()
            .ok_or_else(|| EC2Error::new(format!("No instance reservations for
            {instance_id}")))?
            .instances()
            .first()
            .ok_or_else(|| {

```

```
                EC2Error::new(format!("No instances in reservation for
{instance_id}"))
            }?;

            Ok(instance.clone())
        }

pub async fn start_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Starting instance {instance_id}");

    self.client
        .start_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    tracing::info!("Started instance.");

    Ok(())
}

pub async fn stop_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Stopping instance {instance_id}");

    self.client
        .stop_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    self.wait_for_instance_stopped(instance_id, None).await?;

    tracing::info!("Stopped instance.");

    Ok(())
}

pub async fn reboot_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Rebooting instance {instance_id}");

    self.client
        .reboot_instances()
        .instance_ids(instance_id)
        .send()
```

```

        .await?;

    Ok(())
}

pub async fn wait_for_instance_stopped(
    &self,
    instance_id: &str,
    duration: Option<Duration>,
) -> Result<(), EC2Error> {
    self.client
        .wait_until_instance_stopped()
        .instance_ids(instance_id)
        .wait(duration.unwrap_or(Duration::from_secs(60)))
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to stop.",
                exceeded.max_wait().as_secs(),
            )),
            _ => EC2Error::from(err),
        })?;
    Ok(())
}

pub async fn delete_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Deleting instance with id {instance_id}");
    self.stop_instance(instance_id).await?;
    self.client
        .terminate_instances()
        .instance_ids(instance_id)
        .send()
        .await?;
    self.wait_for_instance_terminated(instance_id).await?;
    tracing::info!("Terminated instance with id {instance_id}");
    Ok(())
}

    async fn wait_for_instance_terminated(&self, instance_id: &str) -> Result<(),
EC2Error> {
    self.client
        .wait_until_instance_terminated()
        .instance_ids(instance_id)
        .wait(Duration::from_secs(60))

```



```
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to terminate.",
                exceeded.max_wait().as_secs(),
            )),
            _ => EC2Error::from(err),
        })?;
    Ok(())
}

pub async fn allocate_ip_address(&self) -> Result<AllocateAddressOutput,
EC2Error> {
    self.client
        .allocate_address()
        .domain(DomainType::Vpc)
        .send()
        .await
        .map_err(EC2Error::from)
}

pub async fn deallocate_ip_address(&self, allocation_id: &str) -> Result<(),
EC2Error> {
    self.client
        .release_address()
        .allocation_id(allocation_id)
        .send()
        .await?;
    Ok(())
}

pub async fn associate_ip_address(
    &self,
    allocation_id: &str,
    instance_id: &str,
) -> Result<AssociateAddressOutput, EC2Error> {
    let response = self
        .client
        .associate_address()
        .allocation_id(allocation_id)
        .instance_id(instance_id)
        .send()
        .await?;
    Ok(response)
}
```

```

    }

    pub async fn disassociate_ip_address(&self, association_id: &str) -> Result<(),
    EC2Error> {
        self.client
            .disassociate_address()
            .association_id(association_id)
            .send()
            .await?;
        Ok(())
    }
}

#[derive(Debug)]
pub struct EC2Error(String);
impl EC2Error {
    pub fn new(value: impl Into<String>) -> Self {
        EC2Error(value.into())
    }

    pub fn add_message(self, message: impl Into<String>) -> Self {
        EC2Error(format!("{}: {}", message.into(), self.0))
    }
}

impl<T: ProvideErrorMetadata> From<T> for EC2Error {
    fn from(value: T) -> Self {
        EC2Error(format!(
            "{}: {}",
            value
                .code()
                .map(String::from)
                .unwrap_or("unknown code".into()),
            value
                .message()
                .map(String::from)
                .unwrap_or("missing reason".into()),
        ))
    }
}

impl std::error::Error for EC2Error {}

impl std::fmt::Display for EC2Error {

```

```

    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}

```

SSM 구조체는 테스트를 위한 오토모크 지점 역할을 하며 해당 함수는 SSM SDK 호출을 래핑합니다.

```

use aws_sdk_ssm::{types::Parameter, Client};
use aws_smithy_async::future::pagination_stream::TryFlatMap;

use crate::ec2::EC2Error;

#[cfg(test)]
use mockall::automock;

#[cfg(not(test))]
pub use SSMImpl as SSM;

#[cfg(test)]
pub use MockSSMImpl as SSM;

pub struct SSMImpl {
    inner: Client,
}

#[cfg_attr(test, automock)]
impl SSMImpl {
    pub fn new(inner: Client) -> Self {
        SSMImpl { inner }
    }

    pub async fn list_path(&self, path: &str) -> Result<Vec<Parameter>, EC2Error> {
        let maybe_params: Vec<Result<Parameter, _>> = TryFlatMap::new(
            self.inner
                .get_parameters_by_path()
                .path(path)
                .into_paginator()
                .send(),
        )
        .flat_map(|item| item.parameters.unwrap_or_default())
    }
}

```

```

        .collect()
        .await;
    // Fail on the first error
    let params = maybe_params
        .into_iter()
        .collect::

```

시나리오는 여러 "관리자" 스타일 구조를 사용하여 시나리오 전체에서 생성 및 삭제된 리소스에 대한 액세스를 처리합니다.

```

use aws_sdk_ec2::operation::{
    allocate_address::AllocateAddressOutput,
    associate_address::AssociateAddressOutput,
};

use crate::ec2::{EC2Error, EC2};

/// ElasticIpManager tracks the lifecycle of a public IP address, including its
/// allocation from the global pool and association with a specific instance.
#[derive(Debug, Default)]
pub struct ElasticIpManager {
    elastic_ip: Option<AllocateAddressOutput>,
    association: Option<AssociateAddressOutput>,
}

impl ElasticIpManager {
    pub fn has_allocation(&self) -> bool {
        self.elastic_ip.is_some()
    }

    pub fn public_ip(&self) -> &str {
        if let Some(allocation) = &self.elastic_ip {
            if let Some(addr) = allocation.public_ip() {
                return addr;
            }
        }
        "0.0.0.0"
    }
}

```

```

pub async fn allocate(&mut self, ec2: &EC2) -> Result<(), EC2Error> {
    let allocation = ec2.allocate_ip_address().await?;
    self.elastic_ip = Some(allocation);
    Ok(())
}

pub async fn associate(&mut self, ec2: &EC2, instance_id: &str) -> Result<(),
EC2Error> {
    if let Some(allocation) = &self.elastic_ip {
        if let Some(allocation_id) = allocation.allocation_id() {
            let association = ec2.associate_ip_address(allocation_id,
instance_id).await?;
            self.association = Some(association);
            return Ok(());
        }
    }
    Err(EC2Error::new("No ip address allocation to associate"))
}

pub async fn remove(mut self, ec2: &EC2) -> Result<(), EC2Error> {
    if let Some(association) = &self.association {
        if let Some(association_id) = association.association_id() {
            ec2.disassociate_ip_address(association_id).await?;
        }
    }
    self.association = None;
    if let Some(allocation) = &self.elastic_ip {
        if let Some(allocation_id) = allocation.allocation_id() {
            ec2.deallocate_ip_address(allocation_id).await?;
        }
    }
    self.elastic_ip = None;
    Ok(())
}
}

use std::fmt::Display;

use aws_sdk_ec2::types::{Instance, InstanceType, KeyPairInfo, SecurityGroup};

use crate::ec2::{EC2Error, EC2};

```

```
/// InstanceManager wraps the lifecycle of an EC2 Instance.
#[derive(Debug, Default)]
pub struct InstanceManager {
    instance: Option<Instance>,
}

impl InstanceManager {
    pub fn instance_id(&self) -> &str {
        if let Some(instance) = &self.instance {
            if let Some(id) = instance.instance_id() {
                return id;
            }
        }
        "Unknown"
    }

    pub fn instance_name(&self) -> &str {
        if let Some(instance) = &self.instance {
            if let Some(tag) = instance.tags().iter().find(|e| e.key() ==
Some("Name")) {
                if let Some(value) = tag.value() {
                    return value;
                }
            }
        }
        "Unknown"
    }

    pub fn instance_ip(&self) -> &str {
        if let Some(instance) = &self.instance {
            if let Some(public_ip_address) = instance.public_ip_address() {
                return public_ip_address;
            }
        }
        "0.0.0.0"
    }

    pub fn instance_display_name(&self) -> String {
        format!("{}", self.instance_name(), self.instance_id())
    }

    /// Create an EC2 instance with the given ID on a given type, using a
    /// generated KeyPair and applying a list of security groups.
    pub async fn create(
```

```
        &mut self,
        ec2: &EC2,
        image_id: &str,
        instance_type: InstanceType,
        key_pair: &KeyPairInfo,
        security_groups: Vec<&SecurityGroup>,
    ) -> Result<(), EC2Error> {
        let instance_id = ec2
            .create_instance(image_id, instance_type, key_pair, security_groups)
            .await?;
        let instance = ec2.describe_instance(&instance_id).await?;
        self.instance = Some(instance);
        Ok(())
    }

    /// Start the managed EC2 instance, if present.
    pub async fn start(&self, ec2: &EC2) -> Result<(), EC2Error> {
        if self.instance.is_some() {
            ec2.start_instance(self.instance_id()).await?;
        }
        Ok(())
    }

    /// Stop the managed EC2 instance, if present.
    pub async fn stop(&self, ec2: &EC2) -> Result<(), EC2Error> {
        if self.instance.is_some() {
            ec2.stop_instance(self.instance_id()).await?;
        }
        Ok(())
    }

    pub async fn reboot(&self, ec2: &EC2) -> Result<(), EC2Error> {
        if self.instance.is_some() {
            ec2.reboot_instance(self.instance_id()).await?;
            ec2.wait_for_instance_stopped(self.instance_id(), None)
                .await?;
            ec2.wait_for_instance_ready(self.instance_id(), None)
                .await?;
        }
        Ok(())
    }

    /// Terminate and delete the managed EC2 instance, if present.
    pub async fn delete(self, ec2: &EC2) -> Result<(), EC2Error> {
```

```
        if self.instance.is_some() {
            ec2.delete_instance(self.instance_id()).await?;
        }
        Ok(())
    }
}

impl Display for InstanceManager {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        if let Some(instance) = &self.instance {
            writeln!(f, "\tID: {}", instance.instance_id().unwrap_or("(Unknown)"))?;
            writeln!(
                f,
                "\tImage ID: {}",
                instance.image_id().unwrap_or("(Unknown)")
            )?;
            writeln!(
                f,
                "\tInstance type: {}",
                instance
                    .instance_type()
                    .map(|it| format!("{it}"))
                    .unwrap_or("(Unknown)".to_string())
            )?;
            writeln!(
                f,
                "\tKey name: {}",
                instance.key_name().unwrap_or("(Unknown)")
            )?;
            writeln!(f, "\tVPC ID: {}", instance.vpc_id().unwrap_or("(Unknown)"))?;
            writeln!(
                f,
                "\tPublic IP: {}",
                instance.public_ip_address().unwrap_or("(Unknown)")
            )?;
            let instance_state = instance
                .state
                .as_ref()
                .map(|is| {
                    is.name()
                        .map(|isn| format!("{isn}"))
                        .unwrap_or("(Unknown)".to_string())
                })
                .unwrap_or("(Unknown)".to_string());
        }
    }
}
```



```
        writeln!(f, "\tState: {instance_state}");
    } else {
        writeln!(f, "\tNo loaded instance");
    }
    Ok(())
}
}

use std::{env, path::PathBuf};

use aws_sdk_ec2::types::KeyPairInfo;

use crate::ec2::{EC2Error, EC2};

use super::util::Util;

/// KeyPairManager tracks a KeyPairInfo and the path the private key has been
/// written to, if it's been created.
#[derive(Debug)]
pub struct KeyPairManager {
    key_pair: KeyPairInfo,
    key_file_path: Option<PathBuf>,
    key_file_dir: PathBuf,
}

impl KeyPairManager {
    pub fn new() -> Self {
        Self::default()
    }

    pub fn key_pair(&self) -> &KeyPairInfo {
        &self.key_pair
    }

    pub fn key_file_path(&self) -> Option<&PathBuf> {
        self.key_file_path.as_ref()
    }

    pub fn key_file_dir(&self) -> &PathBuf {
        &self.key_file_dir
    }

    /// Creates a key pair that can be used to securely connect to an EC2 instance.

```

```

    /// The returned key pair contains private key information that cannot be
    retrieved
    /// again. The private key data is stored as a .pem file.
    ///
    /// :param key_name: The name of the key pair to create.
    pub async fn create(
        &mut self,
        ec2: &EC2,
        util: &Util,
        key_name: String,
    ) -> Result<KeyPairInfo, EC2Error> {
        let (key_pair, material) =
ec2.create_key_pair(key_name.clone()).await.map_err(|e| {
            self.key_pair =
KeyPairInfo::builder().key_name(key_name.clone()).build();
            e.add_message(format!("Couldn't create key {key_name}"))
        })?;

        let path = self.key_file_dir.join(format!("{key_name}.pem"));

        // Save the key_pair information immediately, so it can get cleaned up if
write_secure fails.
        self.key_file_path = Some(path.clone());
        self.key_pair = key_pair.clone();

        util.write_secure(&key_name, &path, material)?;

        Ok(key_pair)
    }

    pub async fn delete(self, ec2: &EC2, util: &Util) -> Result<(), EC2Error> {
        if let Some(key_name) = self.key_pair.key_name() {
            ec2.delete_key_pair(key_name).await?;
            if let Some(key_path) = self.key_file_path() {
                if let Err(err) = util.remove(key_path) {
                    eprintln!("Failed to remove {key_path:?} ({err:?})");
                }
            }
        }
        Ok(())
    }

    pub async fn list(&self, ec2: &EC2) -> Result<Vec<KeyPairInfo>, EC2Error> {
        ec2.list_key_pair().await
    }

```

```

    }
}

impl Default for KeyPairManager {
    fn default() -> Self {
        KeyPairManager {
            key_pair: KeyPairInfo::builder().build(),
            key_file_path: Default::default(),
            key_file_dir: env::temp_dir(),
        }
    }
}

use std::net::Ipv4Addr;

use aws_sdk_ec2::types::SecurityGroup;

use crate::ec2::{EC2Error, EC2};

/// SecurityGroupManager tracks the lifecycle of a SecurityGroup for an instance,
/// including adding a rule to allow SSH from a public IP address.
#[derive(Debug, Default)]
pub struct SecurityGroupManager {
    group_name: String,
    group_description: String,
    security_group: Option<SecurityGroup>,
}

impl SecurityGroupManager {
    pub async fn create(
        &mut self,
        ec2: &EC2,
        group_name: &str,
        group_description: &str,
    ) -> Result<(), EC2Error> {
        self.group_name = group_name.into();
        self.group_description = group_description.into();

        self.security_group = Some(
            ec2.create_security_group(group_name, group_description)
                .await
                .map_err(|e| e.add_message("Couldn't create security group"))?,
        );
    }
}

```

```

        Ok(())
    }

    pub async fn authorize_ingress(&self, ec2: &EC2, ip_address: Ipv4Addr) ->
Result<(), EC2Error> {
    if let Some(sg) = &self.security_group {
        ec2.authorize_security_group_ssh_ingress(
            sg.group_id()
                .ok_or_else(|| EC2Error::new("Missing security group ID"))?,
            vec![ip_address],
        )
        .await?;
    };

    Ok(())
}

pub async fn delete(self, ec2: &EC2) -> Result<(), EC2Error> {
    if let Some(sg) = &self.security_group {
        ec2.delete_security_group(
            sg.group_id()
                .ok_or_else(|| EC2Error::new("Missing security group ID"))?,
        )
        .await?;
    };

    Ok(())
}

pub fn group_name(&self) -> &str {
    &self.group_name
}

pub fn vpc_id(&self) -> Option<&str> {
    self.security_group.as_ref().and_then(|sg| sg.vpc_id())
}

pub fn security_group(&self) -> Option<&SecurityGroup> {
    self.security_group.as_ref()
}
}

impl std::fmt::Display for SecurityGroupManager {

```

```

fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
    match &self.security_group {
        Some(sg) => {
            writeln!(
                f,
                "Security group: {}",
                sg.group_name().unwrap_or("(unknown group)")
            )?;
            writeln!(f, "\tID: {}", sg.group_id().unwrap_or("(unknown group
id)"))?;
            writeln!(f, "\tVPC: {}", sg.vpc_id().unwrap_or("(unknown group
vpc)"))?;
            if !sg.ip_permissions().is_empty() {
                writeln!(f, "\tInbound Permissions:");
                for permission in sg.ip_permissions() {
                    writeln!(f, "\t\t{permission:?}")?;
                }
            }
            Ok(())
        }
        None => writeln!(f, "No security group loaded."),
    }
}
}
}

```

시나리오의 주요 진입점입니다.

```

use ec2_code_examples::{
    ec2::EC2,
    getting_started::{
        scenario::{run, Ec2InstanceScenario},
        util::UtilImpl,
    },
    ssm::SSM,
};

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::load_from_env().await;
    let ec2 = EC2::new(aws_sdk_ec2::Client::new(&sdk_config));
}

```

```
let ssm = SSM::new(aws_sdk_ssm::Client::new(&sdk_config));
let util = UtilImpl {};
let scenario = Ec2InstanceScenario::new(ec2, ssm, util);
run(scenario).await;
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하세요.
  - [AllocateAddress](#)
  - [AssociateAddress](#)
  - [AuthorizeSecurityGroupIngress](#)
  - [CreateKeyPair](#)
  - [CreateSecurityGroup](#)
  - [DeleteKeyPair](#)
  - [DeleteSecurityGroup](#)
  - [DescribeImages](#)
  - [DescribeInstanceTypes](#)
  - [DescribeInstances](#)
  - [DescribeKeyPairs](#)
  - [DescribeSecurityGroups](#)
  - [DisassociateAddress](#)
  - [ReleaseAddress](#)
  - [RunInstances](#)
  - [StartInstances](#)
  - [StopInstances](#)
  - [TerminateInstances](#)
  - [UnmonitorInstances](#)

## 작업

### AllocateAddress

다음 코드 예시에서는 `AllocateAddress`를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn allocate_ip_address(&self) -> Result<AllocateAddressOutput,
EC2Error> {
    self.client
        .allocate_address()
        .domain(DomainType::Vpc)
        .send()
        .await
        .map_err(EC2Error::from)
}
```

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

**AssociateAddress**

다음 코드 예시에서는 AssociateAddress을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn associate_ip_address(
    &self,
    allocation_id: &str,
    instance_id: &str,
) -> Result<AssociateAddressOutput, EC2Error> {
    let response = self
```

```

        .client
        .associate_address()
        .allocation_id(allocation_id)
        .instance_id(instance_id)
        .send()
        .await?;
    Ok(response)
}

```

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

## AuthorizeSecurityGroupIngress

다음 코드 예시에서는 AuthorizeSecurityGroupIngress을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

/// Add an ingress rule to a security group explicitly allowing IPv4 address
/// as {ip}/32 over TCP port 22.
pub async fn authorize_security_group_ssh_ingress(
    &self,
    group_id: &str,
    ingress_ips: Vec<Ipv4Addr>,
) -> Result<(), EC2Error> {
    tracing::info!("Authorizing ingress for security group {group_id}");
    self.client
        .authorize_security_group_ingress()
        .group_id(group_id)
        .set_ip_permissions(Some(
            ingress_ips
                .into_iter()
                .map(|ip| {
                    IpPermission::builder()
                        .ip_protocol("tcp")
                        .from_port(22)
                })
        ))
}

```



```

        .to_port(22)
        .ip_ranges(IpRange::builder().cidr_ip(format!
("{ip}/32")).build())
        .build()
    })
    .collect(),
))
.send()
.await?;
Ok(())
}

```

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

## CreateKeyPair

다음 코드 예시에서는 CreateKeyPair을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

EC2 클라이언트의 create\_key\_pair를 호출하고 반환된 재료를 추출하는 Rust 구현입니다.

```

pub async fn create_key_pair(&self, name: String) -> Result<(KeyPairInfo,
String), EC2Error> {
    tracing::info!("Creating key pair {name}");
    let output = self.client.create_key_pair().key_name(name).send().await?;
    let info = KeyPairInfo::builder()
        .set_key_name(output.key_name)
        .set_key_fingerprint(output.key_fingerprint)
        .set_key_pair_id(output.key_pair_id)
        .build();
    let material = output
        .key_material
        .ok_or_else(|| EC2Error::new("Create Key Pair has no key material"))?;
    Ok((info, material))
}

```

```
}

```

`create_key` impl을 호출하고 PEM 프라이빗 키를 안전하게 저장하는 함수입니다.

```

    /// Creates a key pair that can be used to securely connect to an EC2 instance.
    /// The returned key pair contains private key information that cannot be
retrieved
    /// again. The private key data is stored as a .pem file.
    ///
    /// :param key_name: The name of the key pair to create.
pub async fn create(
    &mut self,
    ec2: &EC2,
    util: &Util,
    key_name: String,
) -> Result<KeyPairInfo, EC2Error> {
    let (key_pair, material) =
ec2.create_key_pair(key_name.clone()).await.map_err(|e| {
        self.key_pair =
KeyPairInfo::builder().key_name(key_name.clone()).build();
        e.add_message(format!("Couldn't create key {key_name}"));
    })?;

    let path = self.key_file_dir.join(format!("{key_name}.pem"));

    // Save the key_pair information immediately, so it can get cleaned up if
write_secure fails.
    self.key_file_path = Some(path.clone());
    self.key_pair = key_pair.clone();

    util.write_secure(&key_name, &path, material)?;

    Ok(key_pair)
}

```

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

## CreateSecurityGroup

다음 코드 예시에서는 `CreateSecurityGroup`을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
pub async fn create_security_group(
    &self,
    name: &str,
    description: &str,
) -> Result<SecurityGroup, EC2Error> {
    tracing::info!("Creating security group {name}");
    let create_output = self
        .client
        .create_security_group()
        .group_name(name)
        .description(description)
        .send()
        .await
        .map_err(EC2Error::from)?;

    let group_id = create_output
        .group_id
        .ok_or_else(|| EC2Error::new("Missing security group id after
creation"))?;

    let group = self
        .describe_security_group(&group_id)
        .await?
        .ok_or_else(|| {
            EC2Error::new(format!("Could not find security group with id
{group_id}"))
        })?;

    tracing::info!("Created security group {name} as {group_id}");

    Ok(group)
}
```

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

## CreateTags

다음 코드 예시에서는 CreateTags을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

이 예제에서는 인스턴스 생성 후 이름 태그를 적용합니다.

```
pub async fn create_instance<'a>(
    &self,
    image_id: &'a str,
    instance_type: InstanceType,
    key_pair: &'a KeyPairInfo,
    security_groups: Vec<&'a SecurityGroup>,
) -> Result<String, EC2Error> {
    let run_instances = self
        .client
        .run_instances()
        .image_id(image_id)
        .instance_type(instance_type)
        .key_name(
            key_pair
                .key_name()
                .ok_or_else(|| EC2Error::new("Missing key name when launching
instance"))?,
        )
        .set_security_group_ids(Some(
            security_groups
                .iter()
                .filter_map(|sg| sg.group_id.clone())
                .collect(),
        ))
        .min_count(1)
        .max_count(1)
```

```
        .send()
        .await?;

    if run_instances.instances().is_empty() {
        return Err(EC2Error::new("Failed to create instance"));
    }

    let instance_id = run_instances.instances()[0].instance_id().unwrap();
    let response = self
        .client
        .create_tags()
        .resources(instance_id)
        .tags(
            Tag::builder()
                .key("Name")
                .value("From SDK Examples")
                .build(),
        )
        .send()
        .await;

    match response {
        Ok(_) => tracing::info!("Created {instance_id} and applied tags."),
        Err(err) => {
            tracing::info!("Error applying tags to {instance_id}: {err:?}");
            return Err(err.into());
        }
    }

    tracing::info!("Instance is created.");


    Ok(instance_id.to_string())
}
```

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

## DeleteKeyPair

다음 코드 예시에서는 DeleteKeyPair을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

백킹 프라이빗 PEM 키도 제거하는 `delete_key` 주위의 래퍼입니다.

```
pub async fn delete(self, ec2: &EC2, util: &Util) -> Result<(), EC2Error> {
    if let Some(key_name) = self.key_pair.key_name() {
        ec2.delete_key_pair(key_name).await?;
        if let Some(key_path) = self.key_file_path() {
            if let Err(err) = util.remove(key_path) {
                eprintln!("Failed to remove {key_path:?} ({err:?})");
            }
        }
    }
    Ok(())
}
```

```
pub async fn delete_key_pair(&self, key_name: &str) -> Result<(), EC2Error> {
    let key_name: String = key_name.into();
    tracing::info!("Deleting key pair {key_name}");
    self.client
        .delete_key_pair()
        .key_name(key_name)
        .send()
        .await?;
    Ok(())
}
```

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

## DeleteSecurityGroup

다음 코드 예시에서는 `DeleteSecurityGroup`을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn delete_security_group(&self, group_id: &str) -> Result<(),
EC2Error> {
    tracing::info!("Deleting security group {group_id}");
    self.client
        .delete_security_group()
        .group_id(group_id)
        .send()
        .await?;
    Ok(())
}
```

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

**DeleteSnapshot**

다음 코드 예시에서는 DeleteSnapshot을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn delete_snapshot(client: &Client, id: &str) -> Result<(), Error> {
    client.delete_snapshot().snapshot_id(id).send().await?;

    println!("Deleted");

    Ok(())
}
```

```
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteSnapshot](#)을 참조하십시오.

## DescribeImages

다음 코드 예시에서는 DescribeImages을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
pub async fn list_images(&self, ids: Vec<Parameter>) -> Result<Vec<Image>,
EC2Error> {
    let image_ids = ids.into_iter().filter_map(|p| p.value).collect();
    let output = self
        .client
        .describe_images()
        .set_image_ids(Some(image_ids))
        .send()
        .await?;

    let images = output.images.unwrap_or_default();
    if images.is_empty() {
        Err(EC2Error::new("No images for selected AMIs"))
    } else {
        Ok(images)
    }
}
```

list\_images 함수를 SSM과 함께 사용하여 환경에 따라를 제한합니다. SSM에 대한 자세한 내용은 [https://docs.aws.amazon.com/systems-manager/latest/userguide/example\\_ssm\\_GetParameters\\_section.html](https://docs.aws.amazon.com/systems-manager/latest/userguide/example_ssm_GetParameters_section.html). 참조하세요.

```
async fn find_image(&mut self) -> Result<ScenarioImage, EC2Error> {
```



```

    let params: Vec<Parameter> = self
        .ssm
        .list_path("/aws/service/ami-amazon-linux-latest")
        .await
        .map_err(|e| e.add_message("Could not find parameters for available
images")))?
        .into_iter()
        .filter(|param| param.name().is_some_and(|name| name.contains("amzn2")))
        .collect();
    let amzn2_images: Vec<ScenarioImage> = self
        .ec2
        .list_images(params)
        .await
        .map_err(|e| e.add_message("Could not find images")))?
        .into_iter()
        .map(ScenarioImage::from)
        .collect();
    println!("We will now create an instance from an Amazon Linux 2 AMI");
    let ami = self.util.select_scenario_image(amzn2_images)?;
    Ok(ami)
}

```

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

## DescribeInstanceStatus

다음 코드 예시에서는 DescribeInstanceStatus을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

async fn show_all_events(client: &Client) -> Result<(), Error> {
    let resp = client.describe_regions().send().await.unwrap();

    for region in resp.regions.unwrap_or_default() {

```

```

let reg: &'static str = Box::leak(Box::from(region.region_name().unwrap()));
let region_provider = RegionProviderChain::default_provider().or_else(reg);
let config = aws_config::from_env().region(region_provider).load().await;
let new_client = Client::new(&config);

let resp = new_client.describe_instance_status().send().await;

println!("Instances in region {}:", reg);
println!();

for status in resp.unwrap().instance_statuses() {
    println!(
        "    Events scheduled for instance ID: {}",
        status.instance_id().unwrap_or_default()
    );
    for event in status.events() {
        println!("    Event ID:      {}",
event.instance_event_id().unwrap());
        println!("    Description:  {}", event.description().unwrap());
        println!("    Event code:   {}", event.code().unwrap().as_ref());
        println!();
    }
}
}

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeInstanceStatus](#)을 참조하십시오.

## DescribeInstanceTypes

다음 코드 예시에서는 DescribeInstanceTypes을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

    /// List instance types that match an image's architecture and are free tier
    eligible.
    pub async fn list_instance_types(&self, image: &Image) ->
    Result<Vec<InstanceType>, EC2Error> {
        let architecture = format!(
            "{}",
            image.architecture().ok_or_else(|| EC2Error::new(format!(
                "Image {:?} does not have a listed architecture",
                image.image_id()
            )))?
        );
        let free_tier_eligible_filter = Filter::builder()
            .name("free-tier-eligible")
            .values("false")
            .build();
        let supported_architecture_filter = Filter::builder()
            .name("processor-info.supported-architecture")
            .values(architecture)
            .build();
        let response = self
            .client
            .describe_instance_types()
            .filters(free_tier_eligible_filter)
            .filters(supported_architecture_filter)
            .send()
            .await?;

        Ok(response
            .instance_types
            .unwrap_or_default()
            .into_iter()
            .filter_map(|iti| iti.instance_type)
            .collect())
    }

```

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

## DescribeInstances

다음 코드 예시에서는 DescribeInstances을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

EC2 인스턴스에 대한 세부 정보를 검색합니다.

```
pub async fn describe_instance(&self, instance_id: &str) -> Result<Instance,
EC2Error> {
    let response = self
        .client
        .describe_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    let instance = response
        .reservations()
        .first()
        .ok_or_else(|| EC2Error::new(format!("No instance reservations for
{instance_id}")))?
        .instances()
        .first()
        .ok_or_else(|| {
            EC2Error::new(format!("No instances in reservation for
{instance_id}"))
        })?;

    Ok(instance.clone())
}
```

EC2 인스턴스를 생성한 후 세부 정보를 검색하고 저장합니다.

```
/// Create an EC2 instance with the given ID on a given type, using a
/// generated KeyPair and applying a list of security groups.
pub async fn create(
    &mut self,
    ec2: &EC2,
```

```

        image_id: &str,
        instance_type: InstanceType,
        key_pair: &KeyPairInfo,
        security_groups: Vec<&SecurityGroup>,
    ) -> Result<(), EC2Error> {
        let instance_id = ec2
            .create_instance(image_id, instance_type, key_pair, security_groups)
            .await?;
        let instance = ec2.describe_instance(&instance_id).await?;
        self.instance = Some(instance);
        Ok(())
    }

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeInstances](#)을 참조하십시오.

## DescribeKeyPairs

다음 코드 예시에서는 DescribeKeyPairs를 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn list_key_pair(&self) -> Result<Vec<KeyPairInfo>, EC2Error> {
    let output = self.client.describe_key_pairs().send().await?;
    Ok(output.key_pairs.unwrap_or_default())
}

```

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

## DescribeRegions

다음 코드 예시에서는 DescribeRegions를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn show_regions(client: &Client) -> Result<(), Error> {
    let rsp = client.describe_regions().send().await?;

    println!("Regions:");
    for region in rsp.regions() {
        println!("  {}", region.region_name().unwrap());
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeRegions](#)을 참조하십시오.

**DescribeSecurityGroups**

다음 코드 예시에서는 DescribeSecurityGroups을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn show_security_groups(client: &aws_sdk_ec2::Client, group_ids: Vec<String>)
{
    let response = client
        .describe_security_groups()
        .set_group_ids(Some(group_ids))
        .send()
```

```

        .await;

    match response {
        Ok(output) => {
            for group in output.security_groups() {
                println!(
                    "Found Security Group {} ({}), vpc id {} and description {}",
                    group.group_name().unwrap_or("unknown"),
                    group.group_id().unwrap_or("id-unknown"),
                    group.vpc_id().unwrap_or("vpcid-unknown"),
                    group.description().unwrap_or("(none)")
                );
            }
        }
        Err(err) => {
            let err = err.into_service_error();
            let meta = err.meta();
            let message = meta.message().unwrap_or("unknown");
            let code = meta.code().unwrap_or("unknown");
            eprintln!("Error listing EC2 Security Groups: ({}code) {}message");
        }
    }
}

```

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

## DescribeSnapshots

다음 코드 예시에서는 DescribeSnapshots을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

스냅샷의 상태를 보여 줍니다.

```

async fn show_state(client: &Client, id: &str) -> Result<(), Error> {

```

```
let resp = client
    .describe_snapshots()
    .filters(Filter::builder().name("snapshot-id").values(id).build())
    .send()
    .await?;

println!(
    "State: {}",
    resp.snapshots().first().unwrap().state().unwrap().as_ref()
);

Ok(())
}
```

```
async fn show_snapshots(client: &Client) -> Result<(), Error> {
    // "self" represents your account ID.
    // You can list the snapshots for any account by replacing
    // "self" with that account ID.
    let resp = client.describe_snapshots().owner_ids("self").send().await?;
    let snapshots = resp.snapshots();
    let length = snapshots.len();

    for snapshot in snapshots {
        println!(
            "ID:          {}",
            snapshot.snapshot_id().unwrap_or_default()
        );
        println!(
            "Description: {}",
            snapshot.description().unwrap_or_default()
        );
        println!("State:          {}", snapshot.state().unwrap().as_ref());
        println!();
    }

    println!();
    println!("Found {} snapshot(s)", length);
    println!();

    Ok(())
}
```



- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeSnapshots](#)을 참조하십시오.

## DisassociateAddress

다음 코드 예시에서는 DisassociateAddress을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn disassociate_ip_address(&self, association_id: &str) -> Result<(),
EC2Error> {
    self.client
        .disassociate_address()
        .association_id(association_id)
        .send()
        .await?;
    Ok(())
}
```

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

## RebootInstances

다음 코드 예시에서는 RebootInstances을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn reboot(&self, ec2: &EC2) -> Result<(), EC2Error> {
```

```

    if self.instance.is_some() {
        ec2.reboot_instance(self.instance_id()).await?;
        ec2.wait_for_instance_stopped(self.instance_id(), None)
            .await?;
        ec2.wait_for_instance_ready(self.instance_id(), None)
            .await?;
    }
    Ok(())
}

```

```

pub async fn reboot_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Rebooting instance {instance_id}");

    self.client
        .reboot_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    Ok(())
}

```

웨이터 API를 사용하여 인스턴스의 웨이터가 중지 및 준비 상태가 됩니다. Waiters API를 사용하려면 rust 파일에 'use aws\_sdk\_ec2::client::Waiters'가 필요합니다.

```

/// Wait for an instance to be ready and status ok (default wait 60 seconds)
pub async fn wait_for_instance_ready(
    &self,
    instance_id: &str,
    duration: Option<Duration>,
) -> Result<(), EC2Error> {
    self.client
        .wait_until_instance_status_ok()
        .instance_ids(instance_id)
        .wait(duration.unwrap_or(Duration::from_secs(60)))
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to start.",
                exceeded.max_wait().as_secs()
            )),
        )),
}

```

```

        _ => EC2Error::from(err),
    })?;
    Ok(())
}

pub async fn wait_for_instance_stopped(
    &self,
    instance_id: &str,
    duration: Option<Duration>,
) -> Result<(), EC2Error> {
    self.client
        .wait_until_instance_stopped()
        .instance_ids(instance_id)
        .wait(duration.unwrap_or(Duration::from_secs(60)))
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to stop.",
                exceeded.max_wait().as_secs(),
            )),
            _ => EC2Error::from(err),
        })?;
    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [RebootInstances](#)을 참조하십시오.

## ReleaseAddress

다음 코드 예시에서는 ReleaseAddress를 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

pub async fn deallocate_ip_address(&self, allocation_id: &str) -> Result<(),
EC2Error> {

```

```

        self.client
            .release_address()
            .allocation_id(allocation_id)
            .send()
            .await?;
        Ok(())
    }

```

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

## RunInstances

다음 코드 예시에서는 RunInstances을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

pub async fn create_instance<'a>(
    &self,
    image_id: &'a str,
    instance_type: InstanceType,
    key_pair: &'a KeyPairInfo,
    security_groups: Vec<&'a SecurityGroup>,
) -> Result<String, EC2Error> {
    let run_instances = self
        .client
        .run_instances()
        .image_id(image_id)
        .instance_type(instance_type)
        .key_name(
            key_pair
                .key_name()
                .ok_or_else(|| EC2Error::new("Missing key name when launching
instance"))?,
        )

```

```

        .set_security_group_ids(Some(
            security_groups
                .iter()
                .filter_map(|sg| sg.group_id.clone())
                .collect(),
        ))
        .min_count(1)
        .max_count(1)
        .send()
        .await?;

if run_instances.instances().is_empty() {
    return Err(EC2Error::new("Failed to create instance"));
}

let instance_id = run_instances.instances()[0].instance_id().unwrap();
let response = self
    .client
    .create_tags()
    .resources(instance_id)
    .tags(
        Tag::builder()
            .key("Name")
            .value("From SDK Examples")
            .build(),
    )
    .send()
    .await;

match response {
    Ok(_) => tracing::info!("Created {instance_id} and applied tags."),
    Err(err) => {
        tracing::info!("Error applying tags to {instance_id}: {err:?}");
        return Err(err.into());
    }
}

tracing::info!("Instance is created.");

Ok(instance_id.to_string())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [RunInstances](#) 참조하세요.

## StartInstances

다음 코드 예시에서는 StartInstances을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

인스턴스 ID로 EC2 인스턴스를 시작합니다.

```
pub async fn start_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Starting instance {instance_id}");

    self.client
        .start_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    tracing::info!("Started instance.");

    Ok(())
}
```

Writers API를 사용하여 인스턴스가 준비 및 상태 정상 상태가 될 때까지 기다립니다. Writers API를 사용하려면 rust 파일에 'use aws\_sdk\_ec2::client::Writers'가 필요합니다.

```
/// Wait for an instance to be ready and status ok (default wait 60 seconds)
pub async fn wait_for_instance_ready(
    &self,
    instance_id: &str,
    duration: Option<Duration>,
) -> Result<(), EC2Error> {
    self.client
        .wait_until_instance_status_ok()
        .instance_ids(instance_id)
```

```

        .wait(duration.unwrap_or(Duration::from_secs(60)))
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to start.",
                exceeded.max_wait().as_secs()
            )),
            _ => EC2Error::from(err),
        })?;
    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [StartInstances](#)을 참조하십시오.

## StopInstances

다음 코드 예시에서는 StopInstances을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

pub async fn stop_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Stopping instance {instance_id}");

    self.client
        .stop_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    self.wait_for_instance_stopped(instance_id, None).await?;

    tracing::info!("Stopped instance.");

    Ok(())
}

```

Writers API를 사용하여 인스턴스가 중지 상태가 될 때까지 기다립니다. Writers API를 사용하려면 rust 파일에 'use aws\_sdk\_ec2::client::Writers'가 필요합니다.

```
pub async fn stop_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Stopping instance {instance_id}");

    self.client
        .stop_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    self.wait_for_instance_stopped(instance_id, None).await?;

    tracing::info!("Stopped instance.");

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [StopInstances](#)을 참조하십시오.

## TerminateInstances

다음 코드 예시에서는 TerminateInstances을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
pub async fn delete_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Deleting instance with id {instance_id}");
    self.stop_instance(instance_id).await?;
    self.client
        .terminate_instances()
```



```

        .instance_ids(instance_id)
        .send()
        .await?;
    self.wait_for_instance_terminated(instance_id).await?;
    tracing::info!("Terminated instance with id {instance_id}");
    Ok(())
}

```

Writers API를 사용하여 인스턴스가 종료 상태가 될 때까지 기다립니다. Writers API를 사용하려면 rust 파일에 'use aws\_sdk\_ec2::client::Writers'가 필요합니다.

```

    async fn wait_for_instance_terminated(&self, instance_id: &str) -> Result<(),
    EC2Error> {
        self.client
            .wait_until_instance_terminated()
            .instance_ids(instance_id)
            .wait(Duration::from_secs(60))
            .await
            .map_err(|err| match err {
                WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                    "Exceeded max time ({}s) waiting for instance to terminate.",
                    exceeded.max_wait().as_secs(),
                )),
                _ => EC2Error::from(err),
            })?;
        Ok(())
    }

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [TerminateInstances](#) 참조하세요.

## SDK for Rust를 사용한 Amazon ECR 예제

다음 코드 예제에서는 Amazon ECR에서 AWS SDK for Rust를 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

### DescribeRepositories

다음 코드 예시에서는 DescribeRepositories를 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn show_repos(client: &aws_sdk_ecr::Client) -> Result<(), aws_sdk_ecr::Error>
{
    let rsp = client.describe_repositories().send().await?;

    let repos = rsp.repositories();

    println!("Found {} repositories:", repos.len());

    for repo in repos {
        println!("  ARN: {}", repo.repository_arn().unwrap());
        println!("  Name: {}", repo.repository_name().unwrap());
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeRepositories](#)를 참조하십시오.

## ListImages

다음 코드 예시에서는 ListImages를 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
async fn show_images(
    client: &aws_sdk_ecr::Client,
    repository: &str,
) -> Result<(), aws_sdk_ecr::Error> {
    let rsp = client
        .list_images()
        .repository_name(repository)
        .send()
        .await?;

    let images = rsp.image_ids();

    println!("found {} images", images.len());

    for image in images {
        println!(
            "image: {}:{}",
            image.image_tag().unwrap(),
            image.image_digest().unwrap()
        );
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListImages](#)를 참조하십시오.

## SDK for Rust를 사용한 Amazon ECS 예제

다음 코드 예제에서는 Amazon ECS에서 AWS SDK for Rust를 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 예시에서는 CreateCluster을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn make_cluster(client: &aws_sdk_ecs::Client, name: &str) -> Result<(),
aws_sdk_ecs::Error> {
    let cluster = client.create_cluster().cluster_name(name).send().await?;
    println!("cluster created: {:?}", cluster);

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateCluster](#)를 참조하십시오.

## DeleteCluster

다음 코드 예시에서는 DeleteCluster을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn remove_cluster(
    client: &aws_sdk_ecs::Client,
    name: &str,
) -> Result<(), aws_sdk_ecs::Error> {
    let cluster_deleted = client.delete_cluster().cluster(name).send().await?;
    println!("cluster deleted: {:?}", cluster_deleted);

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteCluster](#)를 참조하십시오.

## DescribeClusters

다음 코드 예시에서는 DescribeClusters을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn show_clusters(client: &aws_sdk_ecs::Client) -> Result<(),
aws_sdk_ecs::Error> {
    let resp = client.list_clusters().send().await?;
```

```

let cluster_arns = resp.cluster_arns();
println!("Found {} clusters:", cluster_arns.len());

let clusters = client
    .describe_clusters()
    .set_clusters(Some(cluster_arns.into()))
    .send()
    .await?;

for cluster in clusters.clusters() {
    println!("  ARN: {}", cluster.cluster_arn().unwrap());
    println!("  Name: {}", cluster.cluster_name().unwrap());
}

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeClusters](#)를 참조하십시오.

## SDK for Rust를 사용한 Amazon EKS 예제

다음 코드 예제에서는 Amazon EKS에서 AWS SDK for Rust를 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 예시에서는 CreateCluster을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
async fn make_cluster(
    client: &aws_sdk_eks::Client,
    name: &str,
    arn: &str,
    subnet_ids: Vec<String>,
) -> Result<(), aws_sdk_eks::Error> {
    let cluster = client
        .create_cluster()
        .name(name)
        .role_arn(arn)
        .resources_vpc_config(
            VpcConfigRequest::builder()
                .set_subnet_ids(Some(subnet_ids))
                .build(),
        )
        .send()
        .await?;
    println!("cluster created: {:?}", cluster);

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateCluster](#)를 참조하십시오.

## DeleteCluster

다음 코드 예시에서는 DeleteCluster을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn remove_cluster(
    client: &aws_sdk_eks::Client,
    name: &str,
) -> Result<(), aws_sdk_eks::Error> {
    let cluster_deleted = client.delete_cluster().name(name).send().await?;
    println!("cluster deleted: {:?}", cluster_deleted);

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteCluster](#)를 참조하십시오.

## AWS Glue SDK for Rust를 사용한 예제

다음 코드 예제에서는 AWS SDK for Rust를 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.

### 시작

안녕하세요 AWS Glue

다음 코드 예제에서는 AWS Glue의 사용을 시작하는 방법을 보여 줍니다.



## SDK for Rust

**Note**

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

```
let mut list_jobs = glue.list_jobs().into_paginator().send();
while let Some(list_jobs_output) = list_jobs.next().await {
    match list_jobs_output {
        Ok(list_jobs) => {
            let names = list_jobs.job_names();
            info!(?names, "Found these jobs")
        }
        Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListJobs](#)을 참조하십시오.

## 주제

- [기본 사항](#)
- [작업](#)

## 기본 사항

## 기본 사항 알아보기

다음 코드 예제는 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 시작하기](#)를 참조하세요.

## SDK for Rust

### Note

GitHub에 더 많은 내용이 있습니다. [AWS 코드 예시 리포지토리](#)에서 전체 예시를 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.

```
let create_crawler = glue
    .create_crawler()
    .name(self.crawler())
    .database_name(self.database())
    .role(self.iam_role.expose_secret())
    .targets(
        CrawlerTargets::builder()
            .s3_targets(S3Target::builder().path(CRAWLER_TARGET).build())
            .build(),
    )
    .send()
    .await;

match create_crawler {
    Err(err) => {
        let glue_err: aws_sdk_glue::Error = err.into();
        match glue_err {
            aws_sdk_glue::Error::AlreadyExistsException(_) => {
                info!("Using existing crawler");
                Ok(())
            }
            _ => Err(GlueMvpError::GlueSdk(glue_err)),
        }
    }
    Ok(_) => Ok(()),
}??;
```

```

let start_crawler = glue.start_crawler().name(self.crawler()).send().await;

match start_crawler {
    Ok(_) => Ok(()),
    Err(err) => {
        let glue_err: aws_sdk_glue::Error = err.into();
        match glue_err {
            aws_sdk_glue::Error::CrawlerRunningException(_) => Ok(()),
            _ => Err(GlueMvpError::GlueSdk(glue_err)),
        }
    }
}
}?:;

```

의 데이터베이스 및 테이블에 대한 정보를 나열합니다 AWS Glue Data Catalog.

```

let database = glue
    .get_database()
    .name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?
    .to_owned();
let database = database
    .database()
    .ok_or_else(|| GlueMvpError::Unknown("Could not find
database".into()))?;

let tables = glue
    .get_tables()
    .database_name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

let tables = tables.table_list();

```

소스 Amazon S3 버킷에서 CSV 데이터를 추출하고, 필드를 제거하고 이름을 변경하여 변환하고, JSON 형식의 출력을 다른 Amazon S3 버킷으로 로드하는 작업을 만들고 실행합니다.

```

let create_job = glue
    .create_job()

```

```

        .name(self.job())
        .role(self.iam_role.expose_secret())
        .command(
            JobCommand::builder()
                .name("glueetl")
                .python_version("3")
                .script_location(format!("s3://{}/job.py", self.bucket()))
                .build(),
        )
        .glue_version("3.0")
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;

let job_name = create_job.name().ok_or_else(|| {
    GlueMvpError::Unknown("Did not get job name after creating job".into())
})?;

let job_run_output = glue
    .start_job_run()
    .job_name(self.job())
    .arguments("--input_database", self.database())
    .arguments(
        "--input_table",
        self.tables
            .first()
            .ok_or_else(|| GlueMvpError::Unknown("Missing crawler
table".into()))?
            .name(),
    )
    .arguments("--output_bucket_url", self.bucket())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

let job = job_run_output
    .job_run_id()
    .ok_or_else(|| GlueMvpError::Unknown("Missing run id from just started
job".into()))?
    .to_string();

```

데모 중에 생성된 모든 리소스를 삭제합니다.

```
    glue.delete_job()
        .job_name(self.job())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;

    for t in &self.tables {
        glue.delete_table()
            .name(t.name())
            .database_name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
    }

    glue.delete_database()
        .name(self.database())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;

    glue.delete_crawler()
        .name(self.crawler())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;
```

- API 세부 정보는 AWS SDK for Rust 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 예시에서는 CreateCrawler을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
let create_crawler = glue
    .create_crawler()
    .name(self.crawler())
    .database_name(self.database())
    .role(self.iam_role.expose_secret())
    .targets(
        CrawlerTargets::builder()
            .s3_targets(S3Target::builder().path(CRAWLER_TARGET).build())
            .build(),
    )
    .send()
    .await;

match create_crawler {
    Err(err) => {
        let glue_err: aws_sdk_glue::Error = err.into();
        match glue_err {
            aws_sdk_glue::Error::AlreadyExistsException(_) => {
```

```

        info!("Using existing crawler");
        Ok(())
    }
    _ => Err(GlueMvpError::GlueSdk(glue_err)),
}
}
Ok(_) => Ok(()),
}?:;

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateCrawler](#)을 참조하십시오.

## CreateJob

다음 코드 예시에서는 CreateJob을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

let create_job = glue
    .create_job()
    .name(self.job())
    .role(self.iam_role.expose_secret())
    .command(
        JobCommand::builder()
            .name("glueetl")
            .python_version("3")
            .script_location(format!("s3://{}/job.py", self.bucket()))
            .build(),
    )
    .glue_version("3.0")
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

let job_name = create_job.name().ok_or_else(|| {
    GlueMvpError::Unknown("Did not get job name after creating job".into())
});

```

```
})?;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateJob](#)을 참조하십시오.

## DeleteCrawler

다음 코드 예시에서는 DeleteCrawler을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
glue.delete_crawler()
    .name(self.crawler())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteCrawler](#)을 참조하십시오.

## DeleteDatabase

다음 코드 예시에서는 DeleteDatabase을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
glue.delete_database()
```



```

        .name(self.database())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteDatabase](#)을 참조하십시오.

## DeleteJob

다음 코드 예시에서는 DeleteJob을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

glue.delete_job()
    .job_name(self.job())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteJob](#)을 참조하십시오.

## DeleteTable

다음 코드 예시에서는 DeleteTable을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

    for t in &self.tables {
        glue.delete_table()
            .name(t.name())
            .database_name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
    }

```

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

## GetCrawler

다음 코드 예시에서는 GetCrawler을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

let tmp_crawler = glue
    .get_crawler()
    .name(self.crawler())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetCrawler](#)을 참조하십시오.

## GetDatabase

다음 코드 예시에서는 GetDatabase을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
let database = glue
    .get_database()
    .name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?
    .to_owned();
let database = database
    .database()
    .ok_or_else(|| GlueMvpError::Unknown("Could not find
database".into()))?;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetDatabase](#)을 참조하십시오.

**GetJobRun**

다음 코드 예시에서는 GetJobRun을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
let get_job_run = || async {
    Ok:::<JobRun, GlueMvpError>(
        glue.get_job_run()
            .job_name(self.job())
```

```

        .run_id(job_run_id.to_string())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?
        .job_run()
        .ok_or_else(|| GlueMvpError::Unknown("Failed to get
job_run".into()))?
        .to_owned(),
    )
};

let mut job_run = get_job_run().await?;
let mut state =
job_run.job_run_state().unwrap_or(&unknown_state).to_owned();

while matches!(
    state,
    JobRunState::Starting | JobRunState::Stopping | JobRunState::Running
) {
    info!(?state, "Waiting for job to finish");
    tokio::time::sleep(self.wait_delay).await;

    job_run = get_job_run().await?;
    state = job_run.job_run_state().unwrap_or(&unknown_state).to_owned();
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetJobRun](#)을 참조하십시오.

## GetTables

다음 코드 예시에서는 GetTables을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
let tables = glue
```

```

        .get_tables()
        .database_name(self.database())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;

let tables = tables.table_list();

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetTables](#)을 참조하십시오.

## ListJobs

다음 코드 예시에서는 ListJobs을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

let mut list_jobs = glue.list_jobs().into_paginator().send();
while let Some(list_jobs_output) = list_jobs.next().await {
    match list_jobs_output {
        Ok(list_jobs) => {
            let names = list_jobs.job_names();
            info!(?names, "Found these jobs")
        }
        Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
    }
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListJobs](#)을 참조하십시오.

## StartCrawler

다음 코드 예시에서는 StartCrawler을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
let start_crawler = glue.start_crawler().name(self.crawler()).send().await;

match start_crawler {
    Ok(_) => Ok(()),
    Err(err) => {
        let glue_err: aws_sdk_glue::Error = err.into();
        match glue_err {
            aws_sdk_glue::Error::CrawlerRunningException(_) => Ok(()),
            _ => Err(GlueMvpError::GlueSdk(glue_err)),
        }
    }
}
}?:;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [StartCrawler](#)를 참조하십시오.

**StartJobRun**

다음 코드 예시에서는 StartJobRun을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
let job_run_output = glue
    .start_job_run()
    .job_name(self.job())
    .arguments("--input_database", self.database())
```

```

        .arguments(
            "--input_table",
            self.tables
                .first()
                .ok_or_else(|| GlueMvpError::Unknown("Missing crawler
table".into()))?
                .name(),
        )
        .arguments("--output_bucket_url", self.bucket())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;

    let job = job_run_output
        .job_run_id()
        .ok_or_else(|| GlueMvpError::Unknown("Missing run id from just started
job".into()))?
        .to_string();

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [StartJobRun](#)을 참조하십시오.

## SDK for Rust를 사용한 IAM 예제

다음 코드 예제에서는 AWS SDK for Rust를 IAM과 함께 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.

시작

Hello IAM

다음 코드 예제에서는 IAM을 사용하여 시작하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

src/bin/hello.rs에서

```
use aws_sdk_iam::error::SdkError;
use aws_sdk_iam::operation::list_policies::ListPoliciesError;
use clap::Parser;

const PATH_PREFIX_HELP: &str = "The path prefix for filtering the results.";

#[derive(Debug, clap::Parser)]
#[command(about)]
struct HelloScenarioArgs {
    #[arg(long, default_value="/", help=PATH_PREFIX_HELP)]
    pub path_prefix: String,
}

#[tokio::main]
async fn main() -> Result<(), SdkError<ListPoliciesError>> {
    let sdk_config = aws_config::load_from_env().await;
    let client = aws_sdk_iam::Client::new(&sdk_config);

    let args = HelloScenarioArgs::parse();

    iam_service::list_policies(client, args.path_prefix).await?;

    Ok(())
}
```

src/iam-service-lib.rs에서

```
pub async fn list_policies(
    client: iamClient,
    path_prefix: String,
```



```

) -> Result<Vec<String>, SdkError<ListPoliciesError>> {
    let list_policies = client
        .list_policies()
        .path_prefix(path_prefix)
        .scope(PolicyScopeType::Local)
        .into_paginator()
        .items()
        .send()
        .try_collect()
        .await?;

    let policy_names = list_policies
        .into_iter()
        .map(|p| {
            let name = p
                .policy_name
                .unwrap_or_else(|| "Missing Policy Name".to_string());
            println!("{}", name);
            name
        })
        .collect();

    Ok(policy_names)
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListPolicies](#)을 참조하십시오.

## 주제

- [기본 사항](#)
- [작업](#)

## 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 수임할 수 있도록 정책을 추가합니다.
- 역할을 수임하고 임시 보안 인증 정보를 사용하여 S3 버킷을 나열한 후 리소스를 정리합니다.

## SDK for Rust

**i Note**

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

```
use aws_config::meta::region::RegionProviderChain;
use aws_sdk_iam::Error as iamError;
use aws_sdk_iam::{config::Credentials as iamCredentials, config::Region, Client as iamClient};
use aws_sdk_s3::Client as s3Client;
use aws_sdk_sts::Client as stsClient;
use tokio::time::{sleep, Duration};
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), iamError> {
    let (client, uuid, list_all_buckets_policy_document, inline_policy_document) =
        initialize_variables().await;

    if let Err(e) = run_iam_operations(
        client,
        uuid,
        list_all_buckets_policy_document,
```

```

        inline_policy_document,
    )
    .await
    {
        println!("{:?}", e);
    };

    Ok(())
}

async fn initialize_variables() -> (iamClient, String, String, String) {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));

    let shared_config = aws_config::from_env().region(region_provider).load().await;
    let client = iamClient::new(&shared_config);
    let uuid = Uuid::new_v4().to_string();

    let list_all_buckets_policy_document = "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [{
            \"Effect\": \"Allow\",
            \"Action\": \"s3:ListAllMyBuckets\",
            \"Resource\": \"arn:aws:s3:::*\"}]
    }"
    .to_string();
    let inline_policy_document = "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [{
            \"Effect\": \"Allow\",
            \"Action\": \"sts:AssumeRole\",
            \"Resource\": \"{}\"}]
    }"
    .to_string();

    (
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
}

async fn run_iam_operations(
    client: iamClient,

```

```

    uuid: String,
    list_all_buckets_policy_document: String,
    inline_policy_document: String,
) -> Result<(), iamError> {
    let user = iam_service::create_user(&client, &format!("{}", "iam_demo_user_",
uuid)).await?;
    println!("Created the user with the name: {}", user.user_name());
    let key = iam_service::create_access_key(&client, user.user_name()).await?;

    let assume_role_policy_document = "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [{
            \"Effect\": \"Allow\",
            \"Principal\": {\"AWS\": \"{}\"},
            \"Action\": \"sts:AssumeRole\"
        }]
    }"
    .to_string()
    .replace!("{}", user.arn());

    let assume_role_role = iam_service::create_role(
        &client,
        &format!("{}", "iam_demo_role_", uuid),
        &assume_role_policy_document,
    )
    .await?;
    println!("Created the role with the ARN: {}", assume_role_role.arn());

    let list_all_buckets_policy = iam_service::create_policy(
        &client,
        &format!("{}", "iam_demo_policy_", uuid),
        &list_all_buckets_policy_document,
    )
    .await?;
    println!(
        "Created policy: {}",
        list_all_buckets_policy.policy_name.as_ref().unwrap()
    );

    let attach_role_policy_result =
        iam_service::attach_role_policy(&client, &assume_role_role,
&list_all_buckets_policy)
        .await?;
    println!(

```

```

        "Attached the policy to the role: {:?}",
        attach_role_policy_result
    );

    let inline_policy_name = format!("{}", "iam_demo_inline_policy_", uuid);
    let inline_policy_document = inline_policy_document.replace("{}",
assume_role_role.arn());
    iam_service::create_user_policy(&client, &user, &inline_policy_name,
&inline_policy_document)
        .await?;
    println!("Created inline policy.");

    //First, fail to list the buckets with the user.
    let creds = iamCredentials::from_keys(key.access_key_id(),
key.secret_access_key(), None);
    let fail_config = aws_config::from_env()
        .credentials_provider(creds.clone())
        .load()
        .await;
    println!("Fail config: {:?}", fail_config);
    let fail_client: s3Client = s3Client::new(&fail_config);
    match fail_client.list_buckets().send().await {
        Ok(e) => {
            println!("This should not run. {:?}", e);
        }
        Err(e) => {
            println!("Successfully failed with error: {:?}", e)
        }
    }

    let sts_config = aws_config::from_env()
        .credentials_provider(creds.clone())
        .load()
        .await;
    let sts_client: stsClient = stsClient::new(&sts_config);
    sleep(Duration::from_secs(10)).await;
    let assumed_role = sts_client
        .assume_role()
        .role_arn(assume_role_role.arn())
        .role_session_name(format!("iam_demo_assumerole_session_{uuid}"))
        .send()
        .await;
    println!("Assumed role: {:?}", assumed_role);
    sleep(Duration::from_secs(10)).await;

```

```
let assumed_credentials = iamCredentials::from_keys(
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .access_key_id(),
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .secret_access_key(),
    Some(
        assumed_role
            .as_ref()
            .unwrap()
            .credentials
            .as_ref()
            .unwrap()
            .session_token
            .clone(),
    ),
);

let succeed_config = aws_config::from_env()
    .credentials_provider(assumed_credentials)
    .load()
    .await;
println!("succeed config: {:?}", succeed_config);
let succeed_client: s3Client = s3Client::new(&succeed_config);
sleep(Duration::from_secs(10)).await;
match succeed_client.list_buckets().send().await {
    Ok(_) => {
        println!("This should now run successfully.")
    }
    Err(e) => {
        println!("This should not run. {:?}", e);
        panic!()
    }
}
```

```
//Clean up.
iam_service::detach_role_policy(
    &client,
    assume_role_role.role_name(),
    list_all_buckets_policy.arn().unwrap_or_default(),
)
.await?;
iam_service::delete_policy(&client, list_all_buckets_policy).await?;
iam_service::delete_role(&client, &assume_role_role).await?;
println!("Deleted role {}", assume_role_role.role_name());
iam_service::delete_access_key(&client, &user, &key).await?;
println!("Deleted key for {}", key.user_name());
iam_service::delete_user_policy(&client, &user, &inline_policy_name).await?;
println!("Deleted inline user policy: {}", inline_policy_name);
iam_service::delete_user(&client, &user).await?;
println!("Deleted user {}", user.user_name());

Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하십시오.
  - [AttachRolePolicy](#)
  - [CreateAccessKey](#)
  - [CreatePolicy](#)
  - [CreateRole](#)
  - [CreateUser](#)
  - [DeleteAccessKey](#)
  - [DeletePolicy](#)
  - [DeleteRole](#)
  - [DeleteUser](#)
  - [DeleteUserPolicy](#)
  - [DetachRolePolicy](#)
  - [PutUserPolicy](#)

## 작업

### AttachRolePolicy

다음 코드 예시에서는 AttachRolePolicy을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
pub async fn attach_role_policy(
    client: &iamClient,
    role: &Role,
    policy: &Policy,
) -> Result<AttachRolePolicyOutput, SdkError<AttachRolePolicyError>> {
    client
        .attach_role_policy()
        .role_name(role.role_name())
        .policy_arn(policy.arn().unwrap_or_default())
        .send()
        .await
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [AttachRolePolicy](#)을 참조하십시오.

### AttachUserPolicy

다음 코드 예시에서는 AttachUserPolicy을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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



```
pub async fn attach_user_policy(
    client: &iamClient,
    user_name: &str,
    policy_arn: &str,
) -> Result<(), iamError> {
    client
        .attach_user_policy()
        .user_name(user_name)
        .policy_arn(policy_arn)
        .send()
        .await?;

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [AttachUserPolicy](#)을 참조하십시오.

## CreateAccessKey

다음 코드 예시에서는 CreateAccessKey을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
pub async fn create_access_key(client: &iamClient, user_name: &str) ->
    Result<AccessKey, iamError> {
    let mut tries: i32 = 0;
    let max_tries: i32 = 10;

    let response: Result<CreateAccessKeyOutput, SdkError<CreateAccessKeyError>> =
    loop {
        match client.create_access_key().user_name(user_name).send().await {
            Ok(inner_response) => {
                break Ok(inner_response);
            }
        }
    }
}
```

```

        Err(e) => {
            tries += 1;
            if tries > max_tries {
                break Err(e);
            }
            sleep(Duration::from_secs(2)).await;
        }
    };

    Ok(response.unwrap().access_key.unwrap())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateAccessKey](#)을 참조하십시오.

## CreatePolicy

다음 코드 예시에서는 CreatePolicy을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn create_policy(
    client: &iamClient,
    policy_name: &str,
    policy_document: &str,
) -> Result<Policy, iamError> {
    let policy = client
        .create_policy()
        .policy_name(policy_name)
        .policy_document(policy_document)
        .send()
        .await?;
    Ok(policy.policy.unwrap())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreatePolicy](#)을 참조하십시오.

## CreateRole

다음 코드 예시에서는 CreateRole을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn create_role(
    client: &iamClient,
    role_name: &str,
    role_policy_document: &str,
) -> Result<Role, iamError> {
    let response: CreateRoleOutput = loop {
        if let Ok(response) = client
            .create_role()
            .role_name(role_name)
            .assume_role_policy_document(role_policy_document)
            .send()
            .await
        {
            break response;
        }
    };

    Ok(response.role.unwrap())
}
```

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

## CreateServiceLinkedRole

다음 코드 예시에서는 CreateServiceLinkedRole을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn create_service_linked_role(
    client: &iamClient,
    aws_service_name: String,
    custom_suffix: Option<String>,
    description: Option<String>,
) -> Result<CreateServiceLinkedRoleOutput, SdkError<CreateServiceLinkedRoleError>> {
    let response = client
        .create_service_linked_role()
        .aws_service_name(aws_service_name)
        .set_custom_suffix(custom_suffix)
        .set_description(description)
        .send()
        .await?;

    Ok(response)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateServiceLinkedRole](#)을 참조하십시오.

## CreateUser

다음 코드 예시에서는 CreateUser을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn create_user(client: &iamClient, user_name: &str) -> Result<User,
iamError> {
    let response = client.create_user().user_name(user_name).send().await?;

    Ok(response.user.unwrap())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateUser](#)을 참조하십시오.

**DeleteAccessKey**

다음 코드 예시에서는 DeleteAccessKey을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn delete_access_key(
    client: &iamClient,
    user: &User,
    key: &AccessKey,
) -> Result<(), iamError> {
    loop {
        match client
            .delete_access_key()
            .user_name(user.user_name())
```

```

        .access_key_id(key.access_key_id())
        .send()
        .await
    {
        Ok(_) => {
            break;
        }
        Err(e) => {
            println!("Can't delete the access key: {:?}" , e);
            sleep(Duration::from_secs(2)).await;
        }
    }
}
Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteAccessKey](#)을 참조하십시오.

## DeletePolicy

다음 코드 예시에서는 DeletePolicy을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn delete_policy(client: &iamClient, policy: Policy) -> Result<(),
iamError> {
    client
        .delete_policy()
        .policy_arn(policy.arn.unwrap())
        .send()
        .await?;
    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeletePolicy](#)을 참조하십시오.

## DeleteRole

다음 코드 예시에서는 DeleteRole을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn delete_role(client: &iamClient, role: &Role) -> Result<(), iamError> {
    let role = role.clone();
    while client
        .delete_role()
        .role_name(role.role_name())
        .send()
        .await
        .is_err()
    {
        sleep(Duration::from_secs(2)).await;
    }
    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteRole](#)을 참조하십시오.

## DeleteServiceLinkedRole

다음 코드 예시에서는 DeleteServiceLinkedRole을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn delete_service_linked_role(
    client: &iamClient,
    role_name: &str,
) -> Result<(), iamError> {
    client
        .delete_service_linked_role()
        .role_name(role_name)
        .send()
        .await?;

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteServiceLinkedRole](#)을 참조하십시오.

**DeleteUser**

다음 코드 예시에서는 DeleteUser를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn delete_user(client: &iamClient, user: &User) -> Result<(),
    SdkError<DeleteUserError>> {
    let user = user.clone();
```



```

let mut tries: i32 = 0;
let max_tries: i32 = 10;

let response: Result<(), SdkError<DeleteUserError>> = loop {
    match client
        .delete_user()
        .user_name(user.user_name())
        .send()
        .await
    {
        Ok(_) => {
            break Ok(());
        }
        Err(e) => {
            tries += 1;
            if tries > max_tries {
                break Err(e);
            }
            sleep(Duration::from_secs(2)).await;
        }
    }
};

response
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteUser](#)을 참조하십시오.

## DeleteUserPolicy

다음 코드 예시에서는 DeleteUserPolicy을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn delete_user_policy(
```

```

    client: &iamClient,
    user: &User,
    policy_name: &str,
) -> Result<(), SdkError<DeleteUserPolicyError>> {
    client
        .delete_user_policy()
        .user_name(user.user_name())
        .policy_name(policy_name)
        .send()
        .await?;

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteUserPolicy](#)을 참조하십시오.

## DetachRolePolicy

다음 코드 예시에서는 DetachRolePolicy을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn detach_role_policy(
    client: &iamClient,
    role_name: &str,
    policy_arn: &str,
) -> Result<(), iamError> {
    client
        .detach_role_policy()
        .role_name(role_name)
        .policy_arn(policy_arn)
        .send()
        .await?;
}

```

```
    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DetachRolePolicy](#)을 참조하십시오.

## DetachUserPolicy

다음 코드 예시에서는 DetachUserPolicy을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn detach_user_policy(
    client: &iamClient,
    user_name: &str,
    policy_arn: &str,
) -> Result<(), iamError> {
    client
        .detach_user_policy()
        .user_name(user_name)
        .policy_arn(policy_arn)
        .send()
        .await?;

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DetachUserPolicy](#)을 참조하십시오.

## GetAccountPasswordPolicy

다음 코드 예시에서는 GetAccountPasswordPolicy을 사용하는 방법을 보여 줍니다.

## SDK for Rust

### Note

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

```
pub async fn get_account_password_policy(
    client: &iamClient,
) -> Result<GetAccountPasswordPolicyOutput, SdkError<GetAccountPasswordPolicyError>>
{
    let response = client.get_account_password_policy().send().await?;

    Ok(response)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetAccountPasswordPolicy](#)을 참조하십시오.

## GetRole

다음 코드 예시에서는 GetRole을 사용하는 방법을 보여 줍니다.

## SDK for Rust

### Note

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

```
pub async fn get_role(
    client: &iamClient,
    role_name: String,
) -> Result<GetRoleOutput, SdkError<GetRoleError>> {
    let response = client.get_role().role_name(role_name).send().await?;

    Ok(response)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetRole](#)을 참조하십시오.

## ListAttachedRolePolicies

다음 코드 예시에서는 ListAttachedRolePolicies을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn list_attached_role_policies(
    client: &iamClient,
    role_name: String,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListAttachedRolePoliciesOutput, SdkError<ListAttachedRolePoliciesError>>
{
    let response = client
        .list_attached_role_policies()
        .role_name(role_name)
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListAttachedRolePolicies](#)을 참조하십시오.

## ListGroups

다음 코드 예시에서는 ListGroups을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn list_groups(
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListGroupsOutput, SdkError<ListGroupsError>> {
    let response = client
        .list_groups()
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListGroups](#)을 참조하십시오.

**ListPolicies**

다음 코드 예시에서는 ListPolicies를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```

pub async fn list_policies(
    client: iamClient,
    path_prefix: String,
) -> Result<Vec<String>, SdkError<ListPoliciesError>> {
    let list_policies = client
        .list_policies()
        .path_prefix(path_prefix)
        .scope(PolicyScopeType::Local)
        .into_paginator()
        .items()
        .send()
        .try_collect()
        .await?;

    let policy_names = list_policies
        .into_iter()
        .map(|p| {
            let name = p
                .policy_name
                .unwrap_or_else(|| "Missing Policy Name".to_string());
            println!("{}", name);
            name
        })
        .collect();

    Ok(policy_names)
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListPolicies](#)를 참조하십시오.

## ListRolePolicies

다음 코드 예시에서는 ListRolePolicies를 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
pub async fn list_role_policies(
    client: &iamClient,
    role_name: &str,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListRolePoliciesOutput, SdkError<ListRolePoliciesError>> {
    let response = client
        .list_role_policies()
        .role_name(role_name)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListRolePolicies](#)을 참조하십시오.

## ListRoles

다음 코드 예시에서는 ListRoles을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
pub async fn list_roles(
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListRolesOutput, SdkError<ListRolesError>> {
    let response = client
        .list_roles()
        .set_path_prefix(path_prefix)
```



```

        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;
    Ok(response)
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListRoles](#)을 참조하십시오.

## ListSAMLProviders

다음 코드 예시에서는 ListSAMLProviders를 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn list_saml_providers(
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListSAMLProviders](#)을 참조하십시오.

## ListUsers

다음 코드 예시에서는 ListUsers를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn list_users(
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListUsersOutput, SdkError<ListUsersError>> {
    let response = client
        .list_users()
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;
    Ok(response)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListUsers](#)를 참조하십시오.

## AWS IoT SDK for Rust를 사용한 예제

다음 코드 예제에서는 AWS SDK for Rust를 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS IoT.

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

### DescribeEndpoint

다음 코드 예시에서는 DescribeEndpoint을 사용하는 방법을 보여 줍니다.

SDK for Rust

 Note

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

```
async fn show_address(client: &Client, endpoint_type: &str) -> Result<(), Error> {
    let resp = client
        .describe_endpoint()
        .endpoint_type(endpoint_type)
        .send()
        .await?;

    println!("Endpoint address: {}", resp.endpoint_address.unwrap());

    println!();

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeEndpoint](#)를 참조하십시오.

### ListThings

다음 코드 예시에서는 ListThings을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```

async fn show_things(client: &Client) -> Result<(), Error> {
    let resp = client.list_things().send().await?;

    println!("Things:");

    for thing in resp.things.unwrap() {
        println!(
            " Name: {}",
            thing.thing_name.as_deref().unwrap_or_default()
        );
        println!(
            " Type: {}",
            thing.thing_type_name.as_deref().unwrap_or_default()
        );
        println!(
            " ARN:  {}",
            thing.thing_arn.as_deref().unwrap_or_default()
        );
        println!();
    }

    println!();

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListThings](#)을 참조하십시오.

## SDK for Rust를 사용한 Kinesis 예제

다음 코드 예제에서는 Kinesis와 함께 AWS SDK for Rust를 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

### CreateStream

다음 코드 예시에서는 CreateStream을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn make_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client
        .create_stream()
        .stream_name(stream)
        .shard_count(4)
        .send()
        .await?;

    println!("Created stream");

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateStream](#)을 참조하십시오.

## DeleteStream

다음 코드 예시에서는 DeleteStream을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteStream](#)을 참조하십시오.

## DescribeStream

다음 코드 예시에서는 DescribeStream을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn show_stream(client: &Client, stream: &str) -> Result<(), Error> {
    let resp = client.describe_stream().stream_name(stream).send().await?;

    let desc = resp.stream_description.unwrap();
}
```

```

println!("Stream description:");
println!("  Name:           {}: ", desc.stream_name());
println!("  Status:          {:?}" , desc.stream_status());
println!("  Open shards:     {:?}" , desc.shards.len());
println!("  Retention (hours): {}", desc.retention_period_hours());
println!("  Encryption:      {:?}" , desc.encryption_type.unwrap());

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeStream](#)을 참조하십시오.

## ListStreams

다음 코드 예시에서는 ListStreams을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

async fn show_streams(client: &Client) -> Result<(), Error> {
    let resp = client.list_streams().send().await?;

    println!("Stream names:");

    let streams = resp.stream_names;
    for stream in &streams {
        println!("  {}", stream);
    }

    println!("Found {} stream(s)", streams.len());

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListStreams](#)을 참조하십시오.

## PutRecord

다음 코드 예시에서는 PutRecord을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn add_record(client: &Client, stream: &str, key: &str, data: &str) ->
Result<(), Error> {
    let blob = Blob::new(data);

    client
        .put_record()
        .data(blob)
        .partition_key(key)
        .stream_name(stream)
        .send()
        .await?;

    println!("Put data into stream.");

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [PutRecord](#)를 참조하십시오.



## 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 Rust

### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 Kinesis 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error> {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
{}", record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });
}
```

```

    }
  }
});

tracing::info!(
  "Successfully processed {} records",
  event.payload.records.len()
);

Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
  tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    // disable printing the name of the module in every log line.
    .with_target(false)
    // disabling time is handy because CloudWatch will add the ingestion time.
    .without_time()
    .init();

  run(service_fn(function_handler)).await
}

```

## 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 Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 Kinesis 배치 항목 실패를 보고합니다.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
            record.event_id.as_deref().unwrap_or_default()
        );

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* 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 Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    )
}

```

```

    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## AWS KMS SDK for Rust를 사용한 예제

다음 코드 예제에서는 AWS SDK for Rust를 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS KMS.

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

### CreateKey

다음 코드 예시에서는 CreateKey을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn make_key(client: &Client) -> Result<(), Error> {
    let resp = client.create_key().send().await?;

    let id = resp.key_metadata.as_ref().unwrap().key_id();

    println!("Key: {}", id);


    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateKey](#)를 참조하십시오.

### Decrypt

다음 코드 예시에서는 Decrypt을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```

async fn decrypt_key(client: &Client, key: &str, filename: &str) -> Result<(),
Error> {
    // Open input text file and get contents as a string
    // input is a base-64 encoded string, so decode it:
    let data = fs::read_to_string(filename)
        .map(|input| {
            base64::decode(input).expect("Input file does not contain valid base 64
characters.")
        })
        .map(Blob::new);

    let resp = client
        .decrypt()
        .key_id(key)
        .ciphertext_blob(data.unwrap())
        .send()
        .await?;

    let inner = resp.plaintext.unwrap();
    let bytes = inner.as_ref();

    let s = String::from_utf8(bytes.to_vec()).expect("Could not convert to UTF-8");

    println!();
    println!("Decoded string:");
    println!("{}", s);

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [Decrypt](#)를 참조하십시오.

## Encrypt

다음 코드 예시에서는 Encrypt을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn encrypt_string(
    verbose: bool,
    client: &Client,
    text: &str,
    key: &str,
    out_file: &str,
) -> Result<(), Error> {
    let blob = Blob::new(text.as_bytes());

    let resp = client.encrypt().key_id(key).plaintext(blob).send().await?;

    // Did we get an encrypted blob?
    let blob = resp.ciphertext_blob.expect("Could not get encrypted text");
    let bytes = blob.as_ref();

    let s = base64::encode(bytes);

    let mut ofile = File::create(out_file).expect("unable to create file");
    ofile.write_all(s.as_bytes()).expect("unable to write");

    if verbose {
        println!("Wrote the following to {:?}" , out_file);
        println!("{}", s);
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [Encrypt](#)를 참조하십시오.

## GenerateDataKey

다음 코드 예시에서는 GenerateDataKey를 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
async fn make_key(client: &Client, key: &str) -> Result<(), Error> {
    let resp = client
        .generate_data_key()
        .key_id(key)
        .key_spec(DataKeySpec::Aes256)
        .send()
        .await?;

    // Did we get an encrypted blob?
    let blob = resp.ciphertext_blob.expect("Could not get encrypted text");
    let bytes = blob.as_ref();

    let s = base64::encode(bytes);

    println!();
    println!("Data key:");
    println!("{}", s);

    Ok(())
}
```


- API 세부 정보는 AWS SDK for Rust API 참조의 [GenerateDataKey](#)를 참조하십시오.

## GenerateDataKeyWithoutPlaintext

다음 코드 예시에서는 GenerateDataKeyWithoutPlaintext을 사용하는 방법을 보여 줍니다.



## SDK for Rust

 Note

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

```
async fn make_key(client: &Client, key: &str) -> Result<(), Error> {
    let resp = client
        .generate_data_key_without_plaintext()
        .key_id(key)
        .key_spec(DataKeySpec::Aes256)
        .send()
        .await?;

    // Did we get an encrypted blob?
    let blob = resp.ciphertext_blob.expect("Could not get encrypted text");
    let bytes = blob.as_ref();

    let s = base64::encode(bytes);

    println!();
    println!("Data key:");
    println!("{}", s);

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GenerateDataKeyWithoutPlaintext](#)를 참조하십시오.

**GenerateRandom**

다음 코드 예시에서는 GenerateRandom을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
async fn make_string(client: &Client, length: i32) -> Result<(), Error> {
    let resp = client
        .generate_random()
        .number_of_bytes(length)
        .send()
        .await?;

    // Did we get an encrypted blob?
    let blob = resp.plaintext.expect("Could not get encrypted text");
    let bytes = blob.as_ref();

    let s = base64::encode(bytes);

    println!();
    println!("Data key:");
    println!("{}", s);

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GenerateRandom](#)을 참조하십시오.

## ListKeys

다음 코드 예시에서는 ListKeys을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn show_keys(client: &Client) -> Result<(), Error> {
    let resp = client.list_keys().send().await?;

    let keys = resp.keys.unwrap_or_default();

    let len = keys.len();

    for key in keys {
        println!("Key ARN: {}", key.key_arn.as_deref().unwrap_or_default());
    }

    println!();
    println!("Found {} keys", len);

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListKeys](#)를 참조하십시오.

**ReEncrypt**

다음 코드 예시에서는 ReEncrypt을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn reencrypt_string(
    verbose: bool,
    client: &Client,
    input_file: &str,
    output_file: &str,
    first_key: &str,
    new_key: &str,
) -> Result<(), Error> {
    // Get blob from input file
    // Open input text file and get contents as a string
    // input is a base-64 encoded string, so decode it:
    let data = fs::read_to_string(input_file)
        .map(|input_file| base64::decode(input_file).expect("invalid base 64"))
        .map(Blob::new);

    let resp = client
        .re_encrypt()
        .ciphertext_blob(data.unwrap())
        .source_key_id(first_key)
        .destination_key_id(new_key)
        .send()
        .await?;

    // Did we get an encrypted blob?
    let blob = resp.ciphertext_blob.expect("Could not get encrypted text");
    let bytes = blob.as_ref();

    let s = base64::encode(bytes);
    let o = &output_file;

    let mut ofile = File::create(o).expect("unable to create file");
    ofile.write_all(s.as_bytes()).expect("unable to write");

    if verbose {
        println!("Wrote the following to {:}", output_file);
        println!("{}", s);
    } else {
        println!("Wrote base64-encoded output to {}", output_file);
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ReEncrypt](#)를 참조하십시오.

## SDK for Rust를 사용한 Lambda 예제

다음 코드 예제에서는 Lambda와 함께 AWS SDK for Rust를 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.

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

AWS 커뮤니티 기여는 여러 팀이 생성하고 유지 관리하는 예입니다 AWS. 피드백을 제공하려면 연결된 리포지토리에 제공된 메커니즘을 사용합니다.

각 예제에는 컨텍스트에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 전체 소스 코드에 대한 링크가 포함되어 있습니다.

주제

- [기본 사항](#)
- [작업](#)
- [시나리오](#)
- [서버리스 예제](#)
- [AWS 커뮤니티 기여](#)

## 기본 사항

기본 사항 알아보기

다음 코드 예제는 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 참조하십시오.

## SDK for Rust

### Note

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

이 시나리오에 사용된 종속 항목이 있는 Cargo.toml입니다.

```
[package]
name = "lambda-code-examples"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-ec2 = { version = "1.3.0" }
aws-sdk-iam = { version = "1.3.0" }
aws-sdk-lambda = { version = "1.3.0" }
aws-sdk-s3 = { version = "1.4.0" }
aws-smithy-types = { version = "1.0.1" }
aws-types = { version = "1.0.1" }
clap = { version = "4.4", features = ["derive"] }
tokio = { version = "1.20.1", features = ["full"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
tracing = "0.1.37"
serde_json = "1.0.94"
anyhow = "1.0.71"
uuid = { version = "1.3.3", features = ["v4"] }
lambda_runtime = "0.8.0"
serde = "1.0.164"
```

이 시나리오에서 Lambda 직접 호출을 간소화하는 유틸리티 모음입니다. 이 파일은 크레이트에 있는 `src/ations.rs`입니다.

```
use anyhow::anyhow;
use aws_sdk_iam::operation::{create_role::CreateRoleError,
    delete_role::DeleteRoleOutput};
use aws_sdk_lambda::{
    operation::{
        delete_function::DeleteFunctionOutput, get_function::GetFunctionOutput,
        invoke::InvokeOutput, list_functions::ListFunctionsOutput,
        update_function_code::UpdateFunctionCodeOutput,
        update_function_configuration::UpdateFunctionConfigurationOutput,
    },
    primitives::ByteStream,
    types::{Environment, FunctionCode, LastUpdateStatus, State},
};
use aws_sdk_s3::{
    error::ErrorMetadata,
    operation::{delete_bucket::DeleteBucketOutput,
        delete_object::DeleteObjectOutput},
    types::CreateBucketConfiguration,
};
use aws_smithy_types::Blob;
use serde::{ser::SerializeMap, Serialize};
use std::{fmt::Display, path::PathBuf, str::FromStr, time::Duration};
use tracing::{debug, info, warn};

/* Operation describes */
#[derive(Clone, Copy, Debug, Serialize)]
pub enum Operation {
    #[serde(rename = "plus")]
    Plus,
    #[serde(rename = "minus")]
    Minus,
    #[serde(rename = "times")]
    Times,
    #[serde(rename = "divided-by")]
    DividedBy,
}
```

```

impl FromStr for Operation {
    type Err = anyhow::Error;

    fn from_str(s: &str) -> Result<Self, Self::Err> {
        match s {
            "plus" => Ok(Operation::Plus),
            "minus" => Ok(Operation::Minus),
            "times" => Ok(Operation::Times),
            "divided-by" => Ok(Operation::DividedBy),
            _ => Err(anyhow!("Unknown operation {s}")),
        }
    }
}

impl Display for Operation {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            Operation::Plus => write!(f, "plus"),
            Operation::Minus => write!(f, "minus"),
            Operation::Times => write!(f, "times"),
            Operation::DividedBy => write!(f, "divided-by"),
        }
    }
}

/**
 * InvokeArgs will be serialized as JSON and sent to the AWS Lambda handler.
 */
#[derive(Debug)]
pub enum InvokeArgs {
    Increment(i32),
    Arithmetic(Operation, i32, i32),
}

impl Serialize for InvokeArgs {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        match self {
            InvokeArgs::Increment(i) => serializer.serialize_i32(*i),
            InvokeArgs::Arithmetic(o, i, j) => {
                let mut map: S::SerializeMap = serializer.serialize_map(Some(3))?;
                map.serialize_key(&"op".to_string())?;

```



```

        map.serialize_value(&o.to_string())?;
        map.serialize_key(&"i".to_string())?;
        map.serialize_value(&i)?;
        map.serialize_key(&"j".to_string())?;
        map.serialize_value(&j)?;
        map.end()
    }
}
}

/** A policy document allowing Lambda to execute this function on the account's
    behalf. */
const ROLE_POLICY_DOCUMENT: &str = r#{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}";

/**
 * A LambdaManager gathers all the resources necessary to run the Lambda example
 * scenario.
 * This includes instantiated aws_sdk clients and details of resource names.
 */
pub struct LambdaManager {
    iam_client: aws_sdk_iam::Client,
    lambda_client: aws_sdk_lambda::Client,
    s3_client: aws_sdk_s3::Client,
    lambda_name: String,
    role_name: String,
    bucket: String,
    own_bucket: bool,
}

// These unit type structs provide nominal typing on top of String parameters for
// LambdaManager::new
pub struct LambdaName(pub String);
pub struct RoleName(pub String);
pub struct Bucket(pub String);

```

```

pub struct OwnBucket(pub bool);

impl LambdaManager {
    pub fn new(
        iam_client: aws_sdk_iam::Client,
        lambda_client: aws_sdk_lambda::Client,
        s3_client: aws_sdk_s3::Client,
        lambda_name: LambdaName,
        role_name: RoleName,
        bucket: Bucket,
        own_bucket: OwnBucket,
    ) -> Self {
        Self {
            iam_client,
            lambda_client,
            s3_client,
            lambda_name: lambda_name.0,
            role_name: role_name.0,
            bucket: bucket.0,
            own_bucket: own_bucket.0,
        }
    }

    /**
     * Load the AWS configuration from the environment.
     * Look up lambda_name and bucket if none are given, or generate a random name
     if not present in the environment.
     * If the bucket name is provided, the caller needs to have created the bucket.
     * If the bucket name is generated, it will be created.
     */
    pub async fn load_from_env(lambda_name: Option<String>, bucket: Option<String>)
-> Self {
        let sdk_config = aws_config::load_from_env().await;
        let lambda_name = LambdaName(lambda_name.unwrap_or_else(|| {
            std::env::var("LAMBDA_NAME").unwrap_or_else(|_|
"rust_lambda_example".to_string())
        }));
        let role_name = RoleName(format!("{}_role", lambda_name.0));
        let (bucket, own_bucket) =
            match bucket {
                Some(bucket) => (Bucket(bucket), false),
                None => (
                    Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                        format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
                    })

```

```

        })),
        true,
    ),
};

let s3_client = aws_sdk_s3::Client::new(&sdk_config);

if own_bucket {
    info!("Creating bucket for demo: {}", bucket.0);
    s3_client
        .create_bucket()
        .bucket(bucket.0.clone())
        .create_bucket_configuration(
            CreateBucketConfiguration::builder()

.location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                sdk_config.region().unwrap().as_ref(),
            ))
            .build(),
        )
        .send()
        .await
        .unwrap();
}

Self::new(
    aws_sdk_iam::Client::new(&sdk_config),
    aws_sdk_lambda::Client::new(&sdk_config),
    s3_client,
    lambda_name,
    role_name,
    bucket,
    OwnBucket(own_bucket),
)
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
 * The easiest way to create such a zip is to use `cargo lambda build --output-format Zip`.
 */
async fn prepare_function(
    &self,

```

```

        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }

    /**
     * Create a function, uploading from a zip file.
     */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
    anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
            .function_name(self.lambda_name.clone())
            .code(code)
            .role(role.arn())

```

```

        .runtime(aws_sdk_lambda::types::Runtime::Provided12)
        .handler("_unused")
        .send()
        .await
        .map_err( anyhow::Error::from )?;

    self.wait_for_function_ready().await?;

    self.lambda_client
        .publish_version()
        .function_name(self.lambda_name.clone())
        .send()
        .await?;

    Ok(key)
}

/**
 * Create an IAM execution role for the managed Lambda function.
 * If the role already exists, use that instead.
 */
async fn create_role(&self) -> Result<aws_sdk_iam::types::Role, CreateRoleError>
{
    info!("Creating execution role for function");
    let get_role = self
        .iam_client
        .get_role()
        .role_name(self.role_name.clone())
        .send()
        .await;
    if let Ok(get_role) = get_role {
        if let Some(role) = get_role.role {
            return Ok(role);
        }
    }

    let create_role = self
        .iam_client
        .create_role()
        .role_name(self.role_name.clone())
        .assume_role_policy_document(ROLE_POLICY_DOCUMENT)
        .send()
        .await;
}

```

```

    match create_role {
        Ok(create_role) => match create_role.role {
            Some(role) => Ok(role),
            None => Err(CreateRoleError::generic(
                ErrorMetadata::builder()
                    .message("CreateRole returned empty success")
                    .build(),
            )),
        },
        Err(err) => Err(err.into_service_error()),
    }
}

/**
 * Poll `is_function_ready` with a 1-second delay. It returns when the function
 * is ready or when there's an error checking the function's state.
 */
pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
    info!("Waiting for function");
    while !self.is_function_ready(None).await? {
        info!("Function is not ready, sleeping 1s");
        tokio::time::sleep(Duration::from_secs(1)).await;
    }
    Ok(())
}

/**
 * Check if a Lambda function is ready to be invoked.
 * A Lambda function is ready for this scenario when its state is active and its
 * LastUpdateStatus is Successful.
 * Additionally, if a sha256 is provided, the function must have that as its
 * current code hash.
 * Any missing properties or failed requests will be reported as an Err.
 */
async fn is_function_ready(
    &self,
    expected_code_sha256: Option<&str>,
) -> Result<bool, anyhow::Error> {
    match self.get_function().await {
        Ok(func) => {
            if let Some(config) = func.configuration() {
                if let Some(state) = config.state() {
                    info!(?state, "Checking if function is active");
                    if !matches!(state, State::Active) {

```

```

        return Ok(false);
    }
}
match config.last_update_status() {
    Some(last_update_status) => {
        info!(?last_update_status, "Checking if function is
ready");

        match last_update_status {
            LastUpdateStatus::Successful => {
                // continue
            }
            LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                return Ok(false);
            }
            unknown => {
                warn!(
                    status_variant = unknown.as_str(),
                    "LastUpdateStatus unknown"
                );
                return Err(anyhow!(
                    "Unknown LastUpdateStatus, fn config is
{config:?}"
                ));
            }
        }
    }
    None => {
        warn!("Missing last update status");
        return Ok(false);
    }
};
if expected_code_sha256.is_none() {
    return Ok(true);
}
if let Some(code_sha256) = config.code_sha256() {
    return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
}
}
Err(e) => {
    warn!(?e, "Could not get function while waiting");
}
}

```

```
    }
    Ok(false)
}

/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error> {
    info!("Getting lambda function");
    self.lambda_client
        .get_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from)
}

/** List all Lambda functions in the current Region. */
pub async fn list_functions(&self) -> Result<ListFunctionsOutput, anyhow::Error>
{
    info!("Listing lambda functions");
    self.lambda_client
        .list_functions()
        .send()
        .await
        .map_err(anyhow::Error::from)
}

/** Invoke the lambda function using calculator InvokeArgs. */
pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
    info!(?args, "Invoking {}", self.lambda_name);
    let payload = serde_json::to_string(&args)?;
    debug!(?payload, "Sending payload");
    self.lambda_client
        .invoke()
        .function_name(self.lambda_name.clone())
        .payload(Blob::new(payload))
        .send()
        .await
        .map_err(anyhow::Error::from)
}

/** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
pub async fn update_function_code(
```



```

        &self,
        zip_file: PathBuf,
        key: String,
    ) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
        let function_code = self.prepare_function(zip_file, Some(key)).await?;

        info!("Updating code for {}", self.lambda_name);
        let update = self
            .lambda_client
            .update_function_code()
            .function_name(self.lambda_name.clone())
            .s3_bucket(self.bucket.clone())
            .s3_key(function_code.s3_key().unwrap().to_string())
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(update)
    }

    /** Update the environment for a function. */
    pub async fn update_function_configuration(
        &self,
        environment: Environment,
    ) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
        info!(
            ?environment,
            "Updating environment for {}", self.lambda_name
        );
        let updated = self
            .lambda_client
            .update_function_configuration()
            .function_name(self.lambda_name.clone())
            .environment(environment)
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(updated)
    }
}

```

```
/** Delete a function and its role, and if possible or necessary, its associated
code object and bucket. */
pub async fn delete_function(
    &self,
    location: Option<String>,
) -> (
    Result<DeleteFunctionOutput, anyhow::Error>,
    Result<DeleteRoleOutput, anyhow::Error>,
    Option<Result<DeleteObjectOutput, anyhow::Error>>,
) {
    info!("Deleting lambda function {}", self.lambda_name);
    let delete_function = self
        .lambda_client
        .delete_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

    info!("Deleting iam role {}", self.role_name);
    let delete_role = self
        .iam_client
        .delete_role()
        .role_name(self.role_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

    let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
        if let Some(location) = location {
            info!("Deleting object {location}");
            Some(
                self.s3_client
                    .delete_object()
                    .bucket(self.bucket.clone())
                    .key(location)
                    .send()
                    .await
                    .map_err(anyhow::Error::from),
            )
        } else {
            info!(?location, "Skipping delete object");
            None
        }
}
```

```

        };

        (delete_function, delete_role, delete_object)
    }

    pub async fn cleanup(
        &self,
        location: Option<String>,
    ) -> (
        (
            Result<DeleteFunctionOutput, anyhow::Error>,
            Result<DeleteRoleOutput, anyhow::Error>,
            Option<Result<DeleteObjectOutput, anyhow::Error>>,
        ),
        Option<Result<DeleteBucketOutput, anyhow::Error>>,
    ) {
        let delete_function = self.delete_function(location).await;

        let delete_bucket = if self.own_bucket {
            info!("Deleting bucket {}", self.bucket);
            if delete_function.2.is_none() ||
delete_function.2.as_ref().unwrap().is_ok() {
                Some(
                    self.s3_client
                        .delete_bucket()
                        .bucket(self.bucket.clone())
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                None
            }
        } else {
            info!("No bucket to clean up");
            None
        };
    };

    (delete_function, delete_bucket)
}
}

/**

```

```

* Testing occurs primarily as an integration test running the `scenario` bin
successfully.
* Each action relies deeply on the internal workings and state of Amazon Simple
Storage Service (Amazon S3), Lambda, and IAM working together.
* It is therefore infeasible to mock the clients to test the individual actions.
*/
#[cfg(test)]
mod test {
    use super::{InvokeArgs, Operation};
    use serde_json::json;

    /** Make sure that the JSON output of serializing InvokeArgs is what's expected
    by the calculator. */
    #[test]
    fn test_serialize() {
        assert_eq!(json!(InvokeArgs::Increment(5)), 5);
        assert_eq!(
            json!(InvokeArgs::Arithmetic(Operation::Plus, 5, 7)).to_string(),
            r#"{"op":"plus","i":5,"j":7}"#.to_string(),
        );
    }
}

```

일부 동작을 제어하기 위해 명령줄 플래그를 사용하여 시나리오를 처음부터 끝까지 실행하는 바이너리입니다. 이 파일은 크레딧에 있는 `src/bin/scenario.rs`입니다.

```

/*
## Service actions

Service actions wrap the SDK call, taking a client and any specific parameters
necessary for the call.

* CreateFunction
* GetFunction
* ListFunctions
* Invoke
* UpdateFunctionCode
* UpdateFunctionConfiguration
* DeleteFunction

## Scenario

```

A scenario runs at a command prompt and prints output to the user on the result of each service action. A scenario can run in one of two ways: straight through, printing out progress as it goes, or as an interactive question/answer script.

```
## Getting started with functions
```

Use an SDK to manage AWS Lambda functions: create a function, invoke it, update its code, invoke it again, view its output and logs, and delete it.

This scenario uses two Lambda handlers:

Note: Handlers don't use AWS SDK API calls.

The increment handler is straightforward:

1. It accepts a number, increments it, and returns the new value.
2. It performs simple logging of the result.

The arithmetic handler is more complex:

1. It accepts a set of actions ['plus', 'minus', 'times', 'divided-by'] and two numbers, and returns the result of the calculation.
2. It uses an environment variable to control log level (such as DEBUG, INFO, WARNING, ERROR).

It logs a few things at different levels, such as:

- \* DEBUG: Full event data.
- \* INFO: The calculation result.
- \* WARN~ING~: When a divide by zero error occurs.
- \* This will be the typical `RUST\_LOG` variable.

The steps of the scenario are:

1. Create an AWS Identity and Access Management (IAM) role that meets the following requirements:
  - \* Has an `assume_role` policy that grants 'lambda.amazonaws.com' the 'sts:AssumeRole' action.
  - \* Attaches the 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole' managed role.
  - \* You must wait for ~10 seconds after the role is created before you can use it!
2. Create a function (CreateFunction) for the increment handler by packaging it as a zip and doing one of the following:
  - \* Adding it with CreateFunction Code.ZipFile.
  - \* `--or--`

- \* Uploading it to Amazon Simple Storage Service (Amazon S3) and adding it with CreateFunction Code.S3Bucket/S3Key.
  - \* Note: Zipping the file does not have to be done in code.
  - \* If you have a waiter, use it to wait until the function is active. Otherwise, call GetFunction until State is Active.
3. Invoke the function with a number and print the result.
  4. Update the function (UpdateFunctionCode) to the arithmetic handler by packaging it as a zip and doing one of the following:
    - \* Adding it with UpdateFunctionCode ZipFile.
    - \* --or--
    - \* Uploading it to Amazon S3 and adding it with UpdateFunctionCode S3Bucket/S3Key.
  5. Call GetFunction until Configuration.LastUpdateStatus is 'Successful' (or 'Failed').
  6. Update the environment variable by calling UpdateFunctionConfiguration and pass it a log level, such as:
    - \* Environment={'Variables': {'RUST\_LOG': 'TRACE'}}
  7. Invoke the function with an action from the list and a couple of values. Include LogType='Tail' to get logs in the result. Print the result of the calculation and the log.
  8. [Optional] Invoke the function to provoke a divide-by-zero error and show the log result.
  9. List all functions for the account, using pagination (ListFunctions).
  10. Delete the function (DeleteFunction).
  11. Delete the role.

Each step should use the function created in Service Actions to abstract calling the SDK.

```
*/
```

```
use aws_sdk_lambda::{operation::invoke::InvokeOutput, types::Environment};
use clap::Parser;
use std::{collections::HashMap, path::PathBuf};
use tracing::{debug, info, warn};
use tracing_subscriber::EnvFilter;

use lambda_code_examples::actions::{
    InvokeArgs::{Arithmetic, Increment},
    LambdaManager, Operation,
};

#[derive(Debug, Parser)]
pub struct Opt {
    /// The AWS Region.
```

```
#[structopt(short, long)]
pub region: Option<String>,

// The bucket to use for the FunctionCode.
#[structopt(short, long)]
pub bucket: Option<String>,

// The name of the Lambda function.
#[structopt(short, long)]
pub lambda_name: Option<String>,

// The number to increment.
#[structopt(short, long, default_value = "12")]
pub inc: i32,

// The left operand.
#[structopt(long, default_value = "19")]
pub num_a: i32,

// The right operand.
#[structopt(long, default_value = "23")]
pub num_b: i32,

// The arithmetic operation.
#[structopt(short, long, default_value = "plus")]
pub operation: Operation,

#[structopt(long)]
pub cleanup: Option<bool>,

#[structopt(long)]
pub no_cleanup: Option<bool>,
}

fn code_path(lambda: &str) -> PathBuf {
    PathBuf::from(format!("../target/lambda/{lambda}/bootstrap.zip"))
}

fn log_invoke_output(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
}
```

```

    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}

async fn main_block(
    opt: &Opt,
    manager: &LambdaManager,
    code_location: String,
) -> Result<(), anyhow::Error> {
    let invoke = manager.invoke(Increment(opt.inc)).await?;
    log_invoke_output(&invoke, "Invoked function configured as increment");

    let update_code = manager
        .update_function_code(code_path("arithmetic"), code_location.clone())
        .await?;

    let code_sha256 = update_code.code_sha256().unwrap_or("Unknown SHA");
    info!(?code_sha256, "Updated function code with arithmetic.zip");

    let arithmetic_args = Arithmetic(opt.operation, opt.num_a, opt.num_b);
    let invoke = manager.invoke(arithmetic_args).await?;
    log_invoke_output(&invoke, "Invoked function configured as arithmetic");

    let update = manager
        .update_function_configuration(
            Environment::builder()
                .set_variables(Some(HashMap::from([
                    "RUST_LOG".to_string(),
                    "trace".to_string(),
                ])))
                .build(),
        )
        .await?;
    let updated_environment = update.environment();
    info!(?updated_environment, "Updated function configuration");

    let invoke = manager
        .invoke(Arithmetic(opt.operation, opt.num_a, opt.num_b))
        .await?;
    log_invoke_output(

```



```

        &invoke,
        "Invoked function configured as arithmetic with increased logging",
    );

    let invoke = manager
        .invoke(Arithmetic(Operation::DividedBy, opt.num_a, 0))
        .await?;
    log_invoke_output(
        &invoke,
        "Invoked function configured as arithmetic with divide by zero",
    );

    Ok:::<(), anyhow::Error>(( ))
}

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt()
        .without_time()
        .with_file(true)
        .with_line_number(true)
        .with_env_filter(EnvFilter::from_default_env())
        .init();

    let opt = Opt::parse();
    let manager = LambdaManager::load_from_env(opt.lambda_name.clone(),
opt.bucket.clone()).await;

    let key = match manager.create_function(code_path("increment")).await {
        Ok(init) => {
            info!(?init, "Created function, initially with increment.zip");
            let run_block = main_block(&opt, &manager, init.clone()).await;
            info!(?run_block, "Finished running example, cleaning up");
            Some(init)
        }
        Err(err) => {
            warn!(?err, "Error happened when initializing function");
            None
        }
    };

    if Some(false) == opt.cleanup || Some(true) == opt.no_cleanup {
        info!("Skipping cleanup")
    } else {

```

```

        let delete = manager.cleanup(key).await;
        info!(?delete, "Deleted function & cleaned up resources");
    }
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하십시오.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [간접 호출](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## 작업

### CreateFunction

다음 코드 예시에서는 CreateFunction을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```

/**
 * Create a function, uploading from a zip file.
 */
pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
    let code = self.prepare_function(zip_file, None).await?;

    let key = code.s3_key().unwrap().to_string();

```

```

let role = self.create_role().await.map_err(|e| anyhow!(e))?;

info!("Created iam role, waiting 15s for it to become active");
tokio::time::sleep(Duration::from_secs(15)).await;

info!("Creating lambda function {}", self.lambda_name);
let _ = self
    .lambda_client
    .create_function()
    .function_name(self.lambda_name.clone())
    .code(code)
    .role(role.arn())
    .runtime(aws_sdk_lambda::types::Runtime::ProvidedAl2)
    .handler("_unused")
    .send()
    .await
    .map_err(anyhow::Error::from)?;

self.wait_for_function_ready().await?;

self.lambda_client
    .publish_version()
    .function_name(self.lambda_name.clone())
    .send()
    .await?;

Ok(key)
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
 * The easiest way to create such a zip is to use `cargo lambda build --output-
format Zip`.
 */
async fn prepare_function(
    &self,
    zip_file: PathBuf,
    key: Option<String>,
) -> Result<FunctionCode, anyhow::Error> {
    let body = ByteStream::from_path(zip_file).await?;

    let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

```

```

    info!("Uploading function code to s3://{}/{}", self.bucket, key);
    let _ = self
        .s3_client
        .put_object()
        .bucket(self.bucket.clone())
        .key(key.clone())
        .body(body)
        .send()
        .await?;

    Ok(FunctionCode::builder()
        .s3_bucket(self.bucket.clone())
        .s3_key(key)
        .build())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateFunction](#)을 참조하십시오.

## DeleteFunction

다음 코드 예시에서는 DeleteFunction을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

/** Delete a function and its role, and if possible or necessary, its associated
code object and bucket. */
pub async fn delete_function(
    &self,
    location: Option<String>,
) -> (
    Result<DeleteFunctionOutput, anyhow::Error>,
    Result<DeleteRoleOutput, anyhow::Error>,
    Option<Result<DeleteObjectOutput, anyhow::Error>>,
) {
    info!("Deleting lambda function {}", self.lambda_name);
}

```

```

    let delete_function = self
        .lambda_client
        .delete_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

    info!("Deleting iam role {}", self.role_name);
    let delete_role = self
        .iam_client
        .delete_role()
        .role_name(self.role_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

    let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
        if let Some(location) = location {
            info!("Deleting object {location}");
            Some(
                self.s3_client
                    .delete_object()
                    .bucket(self.bucket.clone())
                    .key(location)
                    .send()
                    .await
                    .map_err(anyhow::Error::from),
            )
        } else {
            info!(?location, "Skipping delete object");
            None
        };

    (delete_function, delete_role, delete_object)
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteFunction](#)을 참조하십시오.

## GetFunction

다음 코드 예시에서는 GetFunction을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error> {
    info!("Getting lambda function");
    self.lambda_client
        .get_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetFunction](#)을 참조하십시오.

**Invoke**

다음 코드 예시에서는 Invoke을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
/** Invoke the lambda function using calculator InvokeArgs. */
pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
    info!(?args, "Invoking {}", self.lambda_name);
    let payload = serde_json::to_string(&args)?;
```

```

        debug!(?payload, "Sending payload");
        self.lambda_client
            .invoke()
            .function_name(self.lambda_name.clone())
            .payload(Blob::new(payload))
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

fn log_invoke_output(invoker: &InvokeOutput, message: &str) {
    if let Some(payload) = invoker.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoker.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [간접 호출](#)를 참조하십시오.

## ListFunctions

다음 코드 예시에서는 ListFunctions을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

/** List all Lambda functions in the current Region. */
pub async fn list_functions(&self) -> Result<ListFunctionsOutput, anyhow::Error>
{

```

```

    info!("Listing lambda functions");
    self.lambda_client
        .list_functions()
        .send()
        .await
        .map_err( anyhow::Error::from )
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListFunctions](#)를 참조하십시오.

## UpdateFunctionCode

다음 코드 예시에서는 UpdateFunctionCode을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

/** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
pub async fn update_function_code(
    &self,
    zip_file: PathBuf,
    key: String,
) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
    let function_code = self.prepare_function(zip_file, Some(key)).await?;

    info!("Updating code for {}", self.lambda_name);
    let update = self
        .lambda_client
        .update_function_code()
        .function_name(self.lambda_name.clone())
        .s3_bucket(self.bucket.clone())
        .s3_key(function_code.s3_key().unwrap().to_string())
        .send()
        .await
        .map_err( anyhow::Error::from )?;

```



```

        self.wait_for_function_ready().await?;

        Ok(update)
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --output-format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [UpdateFunctionCode](#)를 참조하십시오.

## UpdateFunctionConfiguration

다음 코드 예시에서는 UpdateFunctionConfiguration을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
/** Update the environment for a function. */
pub async fn update_function_configuration(
    &self,
    environment: Environment,
) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
    info!(
        ?environment,
        "Updating environment for {}", self.lambda_name
    );
    let updated = self
        .lambda_client
        .update_function_configuration()
        .function_name(self.lambda_name.clone())
        .environment(environment)
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(updated)
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [UpdateFunctionConfiguration](#)을 참조하십시오.

## 시나리오

### 사진을 관리하기 위한 서버리스 애플리케이션 만들기

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

## SDK for Rust

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 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- 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 Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda 함수에서 Amazon RDS 데이터베이스에 연결

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
```

```
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
    aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );
};
```

```
    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
SignableBody::Bytes(&[]))
            .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse:::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);
```

```

let pool = sqlx::postgres::PgPoolOptions::new()
    .connect_with(opts)
    .await?;

let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
    .bind(3)
    .bind(2)
    .fetch_one(&pool)
    .await?;

println!("Result: {:?}", result);

Ok(json!({
    "statusCode": 200,
    "content-type": "text/plain",
    "body": format!("The selected sum is: {result}")
}))
}

```

## 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 Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 Kinesis 이벤트를 사용합니다.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

```

```

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error> {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
{}", record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## DynamoDB 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제는 DynamoDB 스트림에서 레코드를 수신하여 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DynamoDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

### SDK for Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features = ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
```



```

        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())
}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}

```

## Amazon DocumentDB 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제는 DocumentDB 변경 스트림에서 레코드를 수신하여 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DocumentDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

## SDK for Rust

**Note**

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

## Rust를 사용하여 Lambda로 Amazon DocumentDB 이벤트 소비

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
    ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(), Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");
```

```

    // Prepare the response
    Ok(())
}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}

```

## Amazon MSK 트리거를 사용하여 Lambda 함수 간접 호출

다음 코드 예제는 Amazon MSK 클러스터에서 레코드를 수신하여 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 MSK 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

### SDK for Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

## Rust를 사용하여 Lambda로 Amazon MSK 이벤트 사용.

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/aws-labs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error> {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

            let record_text = record.value.as_ref().ok_or("Value is None")?;
            info!("Record: {}", &record_text);

            // perform Base64 decoding
            let record_bytes = BASE64_STANDARD.decode(record_text)?;
            let message = std::str::from_utf8(&record_bytes)?;

            info!("Message: {}", message);
        }

    }

    Ok(()).into()
}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```

// required to enable CloudWatch error logging by the runtime
tracing::init_default_subscriber();
info!("Setup CW subscriber!");

run(service_fn(function_handler)).await
}

```

## 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 Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 S3 이벤트를 사용합니다.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust

```

```

    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request from
SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket = evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket name
to exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object key to
exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}

```

## 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 Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 SNS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
// aws_lambda_events = { version = "0.10.0", default-features = false, features = ["sns"] }
// lambda_runtime = "0.8.1"
// tokio = { version = "1", features = ["macros"] }
// tracing = { version = "0.1", features = ["log"] }
// tracing-subscriber = { version = "0.3", default-features = false, features = ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);
}
```

```

    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## 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 Rust

### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 SQS 이벤트를 사용합니다.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
    });
}

```



```

        tracing::info!("Message body: {}",
record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## 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 Rust

### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 Kinesis 배치 항목 실패를 보고합니다.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
}

```

```

    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
            record.event_id.as_deref().unwrap_or_default()
        );

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* 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 Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {

```

```

let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

if let Some(err) = record_data.err() {
    tracing::error!("Error: {}", err);
    return Err(Error::from(err));
}

let record_data = record_data.unwrap_or_default();

// do something interesting with the data
tracing::info!("Data: {}", record_data);

Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## DynamoDB 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제는 DynamoDB 스트림에서 이벤트를 수신하는 Lambda 함수에 대해 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

## SDK for Rust

### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

## Rust를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```

use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifer: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...

```

```

        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* 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 Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        .init();


    run(service_fn(function_handler)).await
}

```

## 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 Rust

 Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 SQS 배치 항목 실패를 보고합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<SqsBatchResponse,
Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

## AWS 커뮤니티 기여

### 서버리스 애플리케이션 빌드 및 테스트

다음 코드 예제는 Lambda 및 DynamoDB와 함께 API Gateway를 사용하여 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

#### SDK for Rust

Rust SDK를 사용하여 Lambda 및 DynamoDB가 포함된 API 게이트웨이로 구성된 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

## SDK for Rust를 사용한 MediaLive 예제

다음 코드 예제에서는 MediaLive에서 AWS SDK for Rust를 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

### ListInputs

다음 코드 예시에서는 ListInputs을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

리전의 MediaLive 입력 이름 및 ARN을 나열하십시오.

```
async fn show_inputs(client: &Client) -> Result<(), Error> {
    let input_list = client.list_inputs().send().await?;

    for i in input_list.inputs() {
        let input_arn = i.arn().unwrap_or_default();
        let input_name = i.name().unwrap_or_default();

        println!("Input Name : {}", input_name);
        println!("Input ARN : {}", input_arn);
        println!();
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListInputs](#)을 참조하십시오.

## SDK for Rust를 사용한 MediaPackage 예제

다음 코드 예제에서는 MediaPackage와 함께 AWS SDK for Rust를 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

### ListChannels

다음 코드 예시에서는 ListChannels를 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

채널 ARN 및 설명을 나열하십시오.

```
async fn show_channels(client: &Client) -> Result<(), Error> {
    let list_channels = client.list_channels().send().await?;

    println!("Channels:");

    for c in list_channels.channels() {
        let description = c.description().unwrap_or_default();
        let arn = c.arn().unwrap_or_default();

        println!(" Description : {}", description);
        println!(" ARN :          {}", arn);
        println!();
    }
}
```

```
    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListChannels](#)을 참조하십시오.

## ListOriginEndpoints

다음 코드 예시에서는 ListOriginEndpoints을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

엔드포인트 설명 및 URL을 나열하십시오.

```
async fn show_endpoints(client: &Client) -> Result<(), Error> {
    let or_endpoints = client.list_origin_endpoints().send().await?;

    println!("Endpoints:");

    for e in or_endpoints.origin_endpoints() {
        let endpoint_url = e.url().unwrap_or_default();
        let endpoint_description = e.description().unwrap_or_default();
        println!(" Description: {}", endpoint_description);
        println!(" URL :      {}", endpoint_url);
        println!();
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListOriginEndpoints](#)를 참조하십시오.

## SDK for Rust를 사용한 Amazon MSK 예제

다음 코드 예제에서는 Amazon MSK에서 AWS SDK for Rust를 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 예제는 Amazon MSK 클러스터에서 레코드를 수신하여 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 MSK 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

SDK for Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 Amazon MSK 이벤트 사용.

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
```

```
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/awslabs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error> {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

            let record_text = record.value.as_ref().ok_or("Value is None")?;
            info!("Record: {}", &record_text);

            // perform Base64 decoding
            let record_bytes = BASE64_STANDARD.decode(record_text)?;
            let message = std::str::from_utf8(&record_bytes)?;

            info!("Message: {}", message);
        }

    }

    Ok(()).into()
}

#[tokio::main]
async fn main() -> Result<(), Error> {

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```

## SDK for Rust를 사용한 Amazon Polly 예제

다음 코드 예제에서는 Amazon Polly에서 AWS SDK for Rust를 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.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 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

#### DescribeVoices

다음 코드 예시에서는 DescribeVoices을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn list_voices(client: &Client) -> Result<(), Error> {
    let resp = client.describe_voices().send().await?;

    println!("Voices:");
}
```

```

let voices = resp.voices();
for voice in voices {
    println!(" Name:      {}", voice.name().unwrap_or("No name!"));
    println!(
        " Language: {}",
        voice.language_name().unwrap_or("No language!")
    );

    println();
}

println!("Found {} voices", voices.len());

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeVoices](#)를 참조하십시오.

## ListLexicons

다음 코드 예시에서는 ListLexicons을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

async fn show_lexicons(client: &Client) -> Result<(), Error> {
    let resp = client.list_lexicons().send().await?;

    println!("Lexicons:");

    let lexicons = resp.lexicons();

    for lexicon in lexicons {
        println!(" Name:      {}", lexicon.name().unwrap_or_default());
        println!(
            " Language: {:?}\n",

```

```

        lexicon
            .attributes()
            .as_ref()
            .map(|attrib| attrib
                .language_code
                .as_ref()
                .expect("languages must have language codes"))
            .expect("languages must have attributes")
    );
}

println!();
println!("Found {} lexicons.", lexicons.len());
println!();

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListLexicons](#)를 참조하십시오.

## PutLexicon

다음 코드 예시에서는 PutLexicon을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

async fn make_lexicon(client: &Client, name: &str, from: &str, to: &str) ->
Result<(), Error> {
    let content = format!("<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/pronunciation-lexicon
\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-lexicon http://
www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\"
alphabet=\"ipa\" xml:lang=\"en-US\">
<lexeme><grapheme>{}</grapheme><alias>{}</alias></lexeme>

```

```

</lexicon>", from, to);

client
    .put_lexicon()
    .name(name)
    .content(content)
    .send()
    .await?;

println!("Added lexicon");

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [PutLexicon](#)을 참조하십시오.

## SynthesizeSpeech

다음 코드 예시에서는 SynthesizeSpeech을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

async fn synthesize(client: &Client, filename: &str) -> Result<(), Error> {
    let content = fs::read_to_string(filename);

    let resp = client
        .synthesize_speech()
        .output_format(OutputFormat::Mp3)
        .text(content.unwrap())
        .voice_id(VoiceId::Joanna)
        .send()
        .await?;

    // Get MP3 data from response and save it
    let mut blob = resp

```



```

        .audio_stream
        .collect()
        .await
        .expect("failed to read data");

let parts: Vec<&str> = filename.split('.').collect();
let out_file = format!("{}", String::from(parts[0]), ".mp3");

let mut file = tokio::fs::File::create(out_file)
    .await
    .expect("failed to create file");

file.write_all_buf(&mut blob)
    .await
    .expect("failed to write to file");

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [SynthesizeSpeech](#)를 참조하십시오.

## 시나리오

텍스트를 스피치로, 다시 스피치에서 텍스트로 변환

다음 코드 예시는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon Polly를 사용하여 일반 텍스트(UTF-8) 입력 파일을 오디오 파일에 합성합니다.
- Amazon S3 버킷에 오디오 파일을 업로드합니다.
- Amazon Transcribe를 사용하여 오디오 파일을 텍스트로 변환합니다.
- 텍스트를 표시합니다.

### SDK for Rust

Amazon Polly를 사용하여 일반 텍스트(UTF-8) 입력 파일을 오디오 파일에 합성하고, 오디오 파일을 Amazon S3 버킷에 업로드하고, Amazon Transcribe를 사용하여 해당 오디오 파일을 텍스트로 변환하고, 텍스트를 표시합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예시를 참조하세요.

이 예시에서 사용되는 서비스

- Amazon Polly
- Amazon S3
- Amazon Transcribe

## SDK for Rust를 사용한 QLDB 예제

다음 코드 예제에서는 QLDB와 함께 AWS SDK for Rust를 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

#### CreateLedger

다음 코드 예시에서는 CreateLedger을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn make_ledger(client: &Client, ledger: &str) -> Result<(), Error> {
    let result = client
        .create_ledger()
```

```

        .name(ledger)
        .permissions_mode(PermissionsMode::AllowAll)
        .send()
        .await?;

println!("ARN: {}", result.arn().unwrap());

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateLedger](#)를 참조하십시오.

## ListLedgers

다음 코드 예시에서는 ListLedgers을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

async fn show_ledgers(client: &QLDBClient) -> Result<(), Error> {
    let mut pages = client.list_ledgers().into_paginator().page_size(2).send();

    while let Some(page) = pages.next().await {
        println!("* {:?}", page); //Prints an entire page of ledgers.
        for ledger in page.unwrap().ledgers() {
            println!("* {:?}", ledger); //Prints the LedgerSummary of a single
ledger.
        }
    }

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListLedgers](#)를 참조하십시오.

# SDK for Rust를 사용한 Amazon RDS 예제

다음 코드 예제에서는 Amazon RDS에서 AWS SDK for Rust를 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

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 Rust

### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda 함수에서 Amazon RDS 데이터베이스에 연결

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};
```

```
const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
SignableBody::Bytes(&[]))
            .expect("signable request");

    let (signing_instructions, _signature) =
```

```

        sign(signable_request, &signing_params.into())?.into_parts());

let mut url = url::Url::parse(&url).unwrap();
for (name, value) in signing_instructions.params() {
    url.query_pairs_mut().append_pair(name, &value);
}

let response = url.to_string().split_off("https://".len());

Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse:::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

    let pool = sqlx::postgres::PgPoolOptions::new()
        .connect_with(opts)
        .await?;

    let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
        .bind(3)
        .bind(2)

```

```
        .fetch_one(&pool)
        .await?;

println!("Result: {:?}", result);

Ok(json!({
    "statusCode": 200,
    "content-type": "text/plain",
    "body": format!("The selected sum is: {result}")
})))
}
```

## SDK for Rust를 사용한 Amazon RDS Data Service 예제

다음 코드 예제에서는 Amazon RDS Data Service에서 AWS SDK for Rust를 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

#### ExecuteStatement

다음 코드 예시에서는 ExecuteStatement을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```

async fn query_cluster(
    client: &Client,
    cluster_arn: &str,
    query: &str,
    secret_arn: &str,
) -> Result<(), Error> {
    let st = client
        .execute_statement()
        .resource_arn(cluster_arn)
        .database("postgres") // Do not confuse this with db instance name
        .sql(query)
        .secret_arn(secret_arn);

    let result = st.send().await?;

    println!("{:?}", result);
    println!();

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ExecuteStatement](#)을 참조하십시오.

## SDK for Rust를 사용한 Amazon Rekognition 예제

다음 코드 예제에서는 Amazon Rekognition과 함께 AWS SDK for Rust를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 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 Rust

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 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

이미지에서 얼굴 감지

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon S3 버킷에 이미지를 저장합니다.
- Amazon Rekognition을 사용하여 연령대, 성별, 감정(예제: 웃음) 등의 얼굴 세부 정보를 감지합니다.

- 이러한 세부 정보를 표시합니다.

## SDK for Rust

uploads 접두사를 사용하여 Amazon S3 버킷에 이미지를 저장하고, Amazon Rekognition을 사용하여 연령대, 성별, 감정(예제: 웃음) 등의 얼굴 세부 정보를 감지한 후 이러한 세부 정보를 표시합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하십시오.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3

## EXIF 및 기타 이미지 정보 저장

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져옵니다.
- Amazon S3 버킷에 이미지 파일을 업로드합니다.
- Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(레이블)을 파악합니다.
- EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

## SDK for Rust

JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져오고, 이미지 파일을 Amazon S3 버킷에 업로드하며, Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(Amazon Rekognition의 레이블)을 파악한 후 EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon Rekognition
- Amazon S3

## SDK for Rust를 사용한 Route 53 예제

다음 코드 예제에서는 Route 53에서 AWS SDK for Rust를 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

#### ListHostedZones

다음 코드 예시에서는 ListHostedZones을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn show_host_info(client: &aws_sdk_route53::Client) -> Result<(),
aws_sdk_route53::Error> {
    let hosted_zone_count = client.get_hosted_zone_count().send().await?;

    println!(
        "Number of hosted zones in region : {}",
        hosted_zone_count.hosted_zone_count(),
    );

    let hosted_zones = client.list_hosted_zones().send().await?;
```

```
println!("Zones:");

for hz in hosted_zones.hosted_zones() {
    let zone_name = hz.name();
    let zone_id = hz.id();

    println!(" ID : {}", zone_id);
    println!(" Name : {}", zone_name);
    println!();
}

Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListHostedZones](#)를 참조하십시오.

## SDK for Rust를 사용한 Amazon S3 예제

다음 코드 예제에서는 Amazon S3에서 AWS SDK for Rust를 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.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 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.

시작하기

Hello Amazon S3

다음 코드 예제에서는 Amazon S3를 사용하여 시작하는 방법을 보여줍니다.

## SDK for Rust

 Note

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

```

/// S3 Hello World Example using the AWS SDK for Rust.
///
/// This example lists the objects in a bucket, uploads an object to that bucket,
/// and then retrieves the object and prints some S3 information about the object.
/// This shows a number of S3 features, including how to use built-in paginators
/// for large data sets.
///
/// # Arguments
///
/// * `client` - an S3 client configured appropriately for the environment.
/// * `bucket` - the bucket name that the object will be uploaded to. Must be
  present in the region the `client` is configured to use.
/// * `filename` - a reference to a path that will be read and uploaded to S3.
/// * `key` - the string key that the object will be uploaded as inside the bucket.
async fn list_bucket_and_upload_object(
    client: &aws_sdk_s3::Client,
    bucket: &str,
    filepath: &Path,
    key: &str,
) -> Result<(), S3ExampleError> {
    // List the buckets in this account
    let mut objects = client
        .list_objects_v2()
        .bucket(bucket)
        .into_paginator()
        .send();

    println!("key\tetag\tlast_modified\tstorage_class");
    while let Some(Ok(object)) = objects.next().await {
        for item in object.contents() {
            println!(
                "{}\t{}\t{}\t{}",
                item.key().unwrap_or_default(),
                item.e_tag().unwrap_or_default(),
            );
        }
    }
}

```

```

        item.last_modified()
            .map(|lm| format!("{lm}"))
            .unwrap_or_default(),
        item.storage_class()
            .map(|sc| format!("{sc}"))
            .unwrap_or_default()
    );
}
}

// Prepare a ByteStream around the file, and upload the object using that
// ByteStream.
let body = aws_sdk_s3::primitives::ByteStream::from_path(filepath)
    .await
    .map_err(|err| {
        S3ExampleError::new(format!(
            "Failed to create bytestream for {filepath:?} ({err:?})"
        ))
    })?;
let resp = client
    .put_object()
    .bucket(bucket)
    .key(key)
    .body(body)
    .send()
    .await?;

println!(
    "Upload success. Version: {:?}",
    resp.version_id()
        .expect("S3 Object upload missing version ID")
);

// Retrieve the just-uploaded object.
let resp = client.get_object().bucket(bucket).key(key).send().await?;
println!("etag: {}", resp.e_tag().unwrap_or("missing"));
println!("version: {}", resp.version_id().unwrap_or("missing"));

Ok(())
}

```

S3ExampleError 유틸리티.

```

/// S3ExampleError provides a From<T: ProvideErrorMetadata> impl to extract
/// client-specific error details. This serves as a consistent backup to handling
/// specific service errors, depending on what is needed by the scenario.
/// It is used throughout the code examples for the AWS SDK for Rust.
#[derive(Debug)]
pub struct S3ExampleError(String);
impl S3ExampleError {
    pub fn new(value: impl Into<String>) -> Self {
        S3ExampleError(value.into())
    }

    pub fn add_message(self, message: impl Into<String>) -> Self {
        S3ExampleError(format!("{}", message.into(), self.0))
    }
}

impl<T: aws_sdk_s3::error::ProvideErrorMetadata> From<T> for S3ExampleError {
    fn from(value: T) -> Self {
        S3ExampleError(format!(
            "{}: {}",
            value
                .code()
                .map(String::from)
                .unwrap_or("unknown code".into()),
            value
                .message()
                .map(String::from)
                .unwrap_or("missing reason".into()),
        ))
    }
}

impl std::error::Error for S3ExampleError {}

impl std::fmt::Display for S3ExampleError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListBuckets](#)을 참조하십시오.

## 주제

- [기본 사항](#)
- [작업](#)
- [시나리오](#)
- [서버리스 예제](#)

## 기본 사항

### 기본 사항 알아보기

다음 코드 예제는 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 Rust

#### Note

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

시나리오를 실행하는 바이너리 크레이트(binary crate)용 코드입니다.

```
#![allow(clippy::result_large_err)]

//! Purpose
//! Shows how to use the AWS SDK for Rust to get started using
//! Amazon Simple Storage Service (Amazon S3). Create a bucket, move objects into
  and out of it,
  //! and delete all resources at the end of the demo.
  //!
```



```

//! This example follows the steps in "Getting started with Amazon S3" in the
    Amazon S3
//! user guide.
//! - https://docs.aws.amazon.com/AmazonS3/latest/userguide/GetStartedWithS3.html

use aws_config::meta::region::RegionProviderChain;
use aws_sdk_s3::{config::Region, Client};
use s3_code_examples::error::S3ExampleError;
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), S3ExampleError> {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
    let region = region_provider.region().await.unwrap();
    let shared_config = aws_config::from_env().region(region_provider).load().await;
    let client = Client::new(&shared_config);
    let bucket_name = format!("amzn-s3-demo-bucket-{}", Uuid::new_v4());
    let file_name = "s3/testfile.txt".to_string();
    let key = "test file key name".to_string();
    let target_key = "target_key".to_string();

    if let Err(e) = run_s3_operations(region, client, bucket_name, file_name, key,
target_key).await
    {
        eprintln!("{:?}", e);
    };

    Ok(())
}

async fn run_s3_operations(
    region: Region,
    client: Client,
    bucket_name: String,
    file_name: String,
    key: String,
    target_key: String,
) -> Result<(), S3ExampleError> {
    s3_code_examples::create_bucket(&client, &bucket_name, &region).await?;
    let run_example: Result<(), S3ExampleError> = (async {
        s3_code_examples::upload_object(&client, &bucket_name, &file_name,
&key).await?;
        let _object = s3_code_examples::download_object(&client, &bucket_name,
&key).await;
    });
}

```

```

        s3_code_examples::copy_object(&client, &bucket_name, &bucket_name, &key,
&target_key)
            .await?;
        s3_code_examples::list_objects(&client, &bucket_name).await?;
        s3_code_examples::clear_bucket(&client, &bucket_name).await?;
        Ok(())
    })
    .await;
    if let Err(err) = run_example {
        eprintln!("Failed to complete getting-started example: {err:?}");
    }
    s3_code_examples::delete_bucket(&client, &bucket_name).await?;

    Ok(())
}

```

시나리오에서 사용되는 일반적인 작업.

```

pub async fn create_bucket(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    region: &aws_config::Region,
) -> Result<Option<aws_sdk_s3::operation::create_bucket::CreateBucketOutput>,
S3ExampleError> {
    let constraint =
aws_sdk_s3::types::BucketLocationConstraint::from(region.to_string().as_str());
    let cfg = aws_sdk_s3::types::CreateBucketConfiguration::builder()
        .location_constraint(constraint)
        .build();
    let create = client
        .create_bucket()
        .create_bucket_configuration(cfg)
        .bucket(bucket_name)
        .send()
        .await;

    // BucketAlreadyExists and BucketAlreadyOwnedByYou are not problems for this
task.
    create.map(Some).or_else(|err| {
        if err
            .as_service_error()

```

```

        .map(|se| se.is_bucket_already_exists() ||
se.is_bucket_already_owned_by_you())
        == Some(true)
        {
            Ok(None)
        } else {
            Err(S3ExampleError::from(err))
        }
    })
}

pub async fn upload_object(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    file_name: &str,
    key: &str,
) -> Result<aws_sdk_s3::operation::put_object::PutObjectOutput, S3ExampleError> {
    let body =
aws_sdk_s3::primitives::ByteStream::from_path(std::path::Path::new(file_name)).await;
    client
        .put_object()
        .bucket(bucket_name)
        .key(key)
        .body(body.unwrap())
        .send()
        .await
        .map_err(S3ExampleError::from)
}

pub async fn download_object(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    key: &str,
) -> Result<aws_sdk_s3::operation::get_object::GetObjectOutput, S3ExampleError> {
    client
        .get_object()
        .bucket(bucket_name)
        .key(key)
        .send()
        .await
        .map_err(S3ExampleError::from)
}

/// Copy an object from one bucket to another.

```

```

pub async fn copy_object(
    client: &aws_sdk_s3::Client,
    source_bucket: &str,
    destination_bucket: &str,
    source_object: &str,
    destination_object: &str,
) -> Result<(), S3ExampleError> {
    let source_key = format!("{source_bucket}/{source_object}");
    let response = client
        .copy_object()
        .copy_source(&source_key)
        .bucket(destination_bucket)
        .key(destination_object)
        .send()
        .await?;

    println!(
        "Copied from {source_key} to {destination_bucket}/{destination_object} with
    etag {}",
        response
            .copy_object_result
            .unwrap_or_else(||
aws_sdk_s3::types::CopyObjectResult::builder().build())
            .e_tag()
            .unwrap_or("missing")
    );
    Ok(())
}

pub async fn list_objects(client: &aws_sdk_s3::Client, bucket: &str) -> Result<(),
S3ExampleError> {
    let mut response = client
        .list_objects_v2()
        .bucket(bucket.to_owned())
        .max_keys(10) // In this example, go 10 at a time.
        .into_paginator()
        .send();

    while let Some(result) = response.next().await {
        match result {
            Ok(output) => {
                for object in output.contents() {
                    println!(" - {}", object.key().unwrap_or("Unknown"));
                }
            }
        }
    }
}

```

```
    }
    Err(err) => {
        eprintln!("{err:?}")
    }
}

Ok(())
}

/// Given a bucket, remove all objects in the bucket, and then ensure no objects
/// remain in the bucket.
pub async fn clear_bucket(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
) -> Result<Vec<String>, S3ExampleError> {
    let objects = client.list_objects_v2().bucket(bucket_name).send().await?;

    // delete_objects no longer needs to be mutable.
    let objects_to_delete: Vec<String> = objects
        .contents()
        .iter()
        .filter_map(|obj| obj.key())
        .map(String::from)
        .collect();

    if objects_to_delete.is_empty() {
        return Ok(vec![]);
    }

    let return_keys = objects_to_delete.clone();

    delete_objects(client, bucket_name, objects_to_delete).await?;

    let objects = client.list_objects_v2().bucket(bucket_name).send().await?;

    eprintln!("{objects:?}");

    match objects.key_count {
        Some(0) => Ok(return_keys),
        _ => Err(S3ExampleError::new(
            "There were still objects left in the bucket.",
        )),
    }
}
```

```
}

pub async fn delete_bucket(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
) -> Result<(), S3ExampleError> {
    let resp = client.delete_bucket().bucket(bucket_name).send().await;
    match resp {
        Ok(_) => Ok(()),
        Err(err) => {
            if err
                .as_service_error()
                .and_then(aws_sdk_s3::error::ProvideErrorMetadata::code)
                == Some("NoSuchBucket")
            {
                Ok(())
            } else {
                Err(S3ExampleError::from(err))
            }
        }
    }
}
```


- API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하십시오.
  - [CopyObject](#)
  - [CreateBucket](#)
  - [DeleteBucket](#)
  - [DeleteObjects](#)
  - [GetObject](#)
  - [ListObjectsV2](#)
  - [PutObject](#)

## 작업

### CompleteMultipartUpload

다음 코드 예시에서는 CompleteMultipartUpload을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
// upload_parts: Vec<aws_sdk_s3::types::CompletedPart>
let completed_multipart_upload: CompletedMultipartUpload =
CompletedMultipartUpload::builder()
    .set_parts(Some(upload_parts))
    .build();

let _complete_multipart_upload_res = client
    .complete_multipart_upload()
    .bucket(&bucket_name)
    .key(&key)
    .multipart_upload(completed_multipart_upload)
    .upload_id(upload_id)
    .send()
    .await?;
```

```
// Create a multipart upload. Use UploadPart and CompleteMultipartUpload to
// upload the file.
let multipart_upload_res: CreateMultipartUploadOutput = client
    .create_multipart_upload()
    .bucket(&bucket_name)
    .key(&key)
    .send()
    .await?;

let upload_id = multipart_upload_res.upload_id().ok_or(S3ExampleError::new(
    "Missing upload_id after CreateMultipartUpload",
    ))?;
```

```
let mut upload_parts: Vec<aws_sdk_s3::types::CompletedPart> = Vec::new();

for chunk_index in 0..chunk_count {
```

```
    let this_chunk = if chunk_count - 1 == chunk_index {
        size_of_last_chunk
    } else {
        CHUNK_SIZE
    };
    let stream = ByteStream::read_from()
        .path(path)
        .offset(chunk_index * CHUNK_SIZE)
        .length(Length::Exact(this_chunk))
        .build()
        .await
        .unwrap();

    // Chunk index needs to start at 0, but part numbers start at 1.
    let part_number = (chunk_index as i32) + 1;
    let upload_part_res = client
        .upload_part()
        .key(&key)
        .bucket(&bucket_name)
        .upload_id(upload_id)
        .body(stream)
        .part_number(part_number)
        .send()
        .await?;

    upload_parts.push(
        CompletedPart::builder()
            .e_tag(upload_part_res.e_tag.unwrap_or_default())
            .part_number(part_number)
            .build(),
    );
}
```


- API 세부 정보는 AWS SDK for Rust API 참조의 [CompleteMultipartUpload](#)를 참조하십시오.

## CopyObject

다음 코드 예시에서는 CopyObject를 사용하는 방법을 보여 줍니다.



## SDK for Rust

 Note

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

```
/// Copy an object from one bucket to another.
pub async fn copy_object(
    client: &aws_sdk_s3::Client,
    source_bucket: &str,
    destination_bucket: &str,
    source_object: &str,
    destination_object: &str,
) -> Result<(), S3ExampleError> {
    let source_key = format!("{source_bucket}/{source_object}");
    let response = client
        .copy_object()
        .copy_source(&source_key)
        .bucket(destination_bucket)
        .key(destination_object)
        .send()
        .await?;

    println!(
        "Copied from {source_key} to {destination_bucket}/{destination_object} with
    etag {}",
        response
            .copy_object_result
            .unwrap_or_else(||
aws_sdk_s3::types::CopyObjectResult::builder().build())
            .e_tag()
            .unwrap_or("missing")
    );
    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CopyObject](#)을 참조하십시오.

## CreateBucket

다음 코드 예시에서는 CreateBucket을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```
pub async fn create_bucket(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    region: &aws_config::Region,
) -> Result<Option<aws_sdk_s3::operation::create_bucket::CreateBucketOutput>,
S3ExampleError> {
    let constraint =
aws_sdk_s3::types::BucketLocationConstraint::from(region.to_string().as_str());
    let cfg = aws_sdk_s3::types::CreateBucketConfiguration::builder()
        .location_constraint(constraint)
        .build();
    let create = client
        .create_bucket()
        .create_bucket_configuration(cfg)
        .bucket(bucket_name)
        .send()
        .await;

    // BucketAlreadyExists and BucketAlreadyOwnedByYou are not problems for this
    task.
    create.map(Some).or_else(|err| {
        if err
            .as_service_error()
            .map(|se| se.is_bucket_already_exists() ||
se.is_bucket_already_owned_by_you())
            == Some(true)
        {
            Ok(None)
        } else {
            Err(S3ExampleError::from(err))
        }
    })
}
```

```
    })
}
```

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

## CreateMultipartUpload

다음 코드 예시에서는 CreateMultipartUpload을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
// Create a multipart upload. Use UploadPart and CompleteMultipartUpload to
// upload the file.
let multipart_upload_res: CreateMultipartUploadOutput = client
    .create_multipart_upload()
    .bucket(&bucket_name)
    .key(&key)
    .send()
    .await?;

let upload_id = multipart_upload_res.upload_id().ok_or(S3ExampleError::new(
    "Missing upload_id after CreateMultipartUpload",
))?;
```

```
let mut upload_parts: Vec<aws_sdk_s3::types::CompletedPart> = Vec::new();

for chunk_index in 0..chunk_count {
    let this_chunk = if chunk_count - 1 == chunk_index {
        size_of_last_chunk
    } else {
        CHUNK_SIZE
    };
    let stream = ByteStream::read_from()
```

```
        .path(path)
        .offset(chunk_index * CHUNK_SIZE)
        .length(Length::Exact(this_chunk))
        .build()
        .await
        .unwrap();

// Chunk index needs to start at 0, but part numbers start at 1.
let part_number = (chunk_index as i32) + 1;
let upload_part_res = client
    .upload_part()
    .key(&key)
    .bucket(&bucket_name)
    .upload_id(upload_id)
    .body(stream)
    .part_number(part_number)
    .send()
    .await?;

upload_parts.push(
    CompletedPart::builder()
        .e_tag(upload_part_res.e_tag.unwrap_or_default())
        .part_number(part_number)
        .build(),
);
}
```

```
// upload_parts: Vec<aws_sdk_s3::types::CompletedPart>
let completed_multipart_upload: CompletedMultipartUpload =
CompletedMultipartUpload::builder()
    .set_parts(Some(upload_parts))
    .build();

let _complete_multipart_upload_res = client
    .complete_multipart_upload()
    .bucket(&bucket_name)
    .key(&key)
    .multipart_upload(completed_multipart_upload)
    .upload_id(upload_id)
    .send()
    .await?;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateMultipartUpload](#)를 참조하십시오.

## DeleteBucket

다음 코드 예시에서는 DeleteBucket을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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


```
pub async fn delete_bucket(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
) -> Result<(), S3ExampleError> {
    let resp = client.delete_bucket().bucket(bucket_name).send().await;
    match resp {
        Ok(_) => Ok(()),
        Err(err) => {
            if err
                .as_service_error()
                .and_then(aws_sdk_s3::error::ProvideErrorMetadata::code)
                == Some("NoSuchBucket")
            {
                Ok(())
            } else {
                Err(S3ExampleError::from(err))
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteBucket](#)을 참조하십시오.

## DeleteObject

다음 코드 예시에서는 DeleteObject을 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```
/// Delete an object from a bucket.
pub async fn remove_object(
    client: &aws_sdk_s3::Client,
    bucket: &str,
    key: &str,
) -> Result<(), S3ExampleError> {
    client
        .delete_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await?;

    // There are no modeled errors to handle when deleting an object.


    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteObject](#)을 참조하십시오.

## DeleteObjects

다음 코드 예시에서는 DeleteObjects를 사용하는 방법을 보여 줍니다.

## SDK for Rust

 Note

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

```

/// Delete the objects in a bucket.
pub async fn delete_objects(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    objects_to_delete: Vec<String>,
) -> Result<(), S3ExampleError> {
    // Push into a mut vector to use `?` early return errors while building object
    keys.
    let mut delete_object_ids: Vec<aws_sdk_s3::types::ObjectIdentifier> = vec![];
    for obj in objects_to_delete {
        let obj_id = aws_sdk_s3::types::ObjectIdentifier::builder()
            .key(obj)
            .build()
            .map_err(|err| {
                S3ExampleError::new(format!("Failed to build key for delete_object:
{err:?}"))
            })?;
        delete_object_ids.push(obj_id);
    }

    client
        .delete_objects()
        .bucket(bucket_name)
        .delete(
            aws_sdk_s3::types::Delete::builder()
                .set_objects(Some(delete_object_ids))
                .build()
                .map_err(|err| {
                    S3ExampleError::new(format!("Failed to build delete_object input
{err:?}"))
                })?,
        )
        .send()
        .await?;
    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DeleteObjects](#)을 참조하십시오.

## GetBucketLocation

다음 코드 예시에서는 `GetBucketLocation`을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
async fn show_buckets(
    strict: bool,
    client: &Client,
    region: BucketLocationConstraint,
) -> Result<(), S3ExampleError> {
    let mut buckets = client.list_buckets().into_paginator().send();

    let mut num_buckets = 0;
    let mut in_region = 0;

    while let Some(Ok(output)) = buckets.next().await {
        for bucket in output.buckets() {
            num_buckets += 1;
            if strict {
                let r = client
                    .get_bucket_location()
                    .bucket(bucket.name().unwrap_or_default())
                    .send()
                    .await?;

                if r.location_constraint() == Some(&region) {
                    println!("{}", bucket.name().unwrap_or_default());
                    in_region += 1;
                }
            } else {
                println!("{}", bucket.name().unwrap_or_default());
            }
        }
    }

    println!();
}
```



```

    if strict {
        println!(
            "Found {} buckets in the {} region out of a total of {} buckets.",
            in_region, region, num_buckets
        );
    } else {
        println!("Found {} buckets in all regions.", num_buckets);
    }

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetBucketLocation](#)을 참조하십시오.

## GetObject

다음 코드 예시에서는 GetObject을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

async fn get_object(client: Client, opt: Opt) -> Result<usize, S3ExampleError> {
    trace!("bucket:      {}", opt.bucket);
    trace!("object:       {}", opt.object);
    trace!("destination: {}", opt.destination.display());

    let mut file = File::create(opt.destination.clone()).map_err(|err| {
        S3ExampleError::new(format!(
            "Failed to initialize file for saving S3 download: {err:?}"
        ))
    })?;

    let mut object = client
        .get_object()
        .bucket(opt.bucket)
        .key(opt.object)

```

```

        .send()
        .await?;

    let mut byte_count = 0_usize;
    while let Some(bytes) = object.body.try_next().await.map_err(|err| {
        S3ExampleError::new(format!("Failed to read from S3 download stream:
{err:?}"))
    })? {
        let bytes_len = bytes.len();
        file.write_all(&bytes).map_err(|err| {
            S3ExampleError::new(format!(
                "Failed to write from S3 download stream to local file: {err:?}")
            ))
        })?;
        trace!("Intermediate write of {bytes_len}");
        byte_count += bytes_len;
    }

    Ok(byte_count)
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetObject](#)을 참조하십시오.

## ListBuckets

다음 코드 예시에서는 ListBuckets을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

async fn show_buckets(
    strict: bool,
    client: &Client,
    region: BucketLocationConstraint,
) -> Result<(), S3ExampleError> {
    let mut buckets = client.list_buckets().into_paginator().send();

```

```

let mut num_buckets = 0;
let mut in_region = 0;

while let Some(Ok(output)) = buckets.next().await {
    for bucket in output.buckets() {
        num_buckets += 1;
        if strict {
            let r = client
                .get_bucket_location()
                .bucket(bucket.name().unwrap_or_default())
                .send()
                .await?;

            if r.location_constraint() == Some(&region) {
                println!("{}", bucket.name().unwrap_or_default());
                in_region += 1;
            }
        } else {
            println!("{}", bucket.name().unwrap_or_default());
        }
    }
}

println!();
if strict {
    println!(
        "Found {} buckets in the {} region out of a total of {} buckets.",
        in_region, region, num_buckets
    );
} else {
    println!("Found {} buckets in all regions.", num_buckets);
}

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListBuckets](#)을 참조하십시오.

## ListObjectVersions

다음 코드 예시에서는 ListObjectVersions을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn show_versions(client: &Client, bucket: &str) -> Result<(), Error> {
    let resp = client.list_object_versions().bucket(bucket).send().await?;

    for version in resp.versions() {
        println!("{}", version.key().unwrap_or_default());
        println!(" version ID: {}", version.version_id().unwrap_or_default());
        println!();
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListObjectVersions](#)을 참조하십시오.

**ListObjectsV2**

다음 코드 예시에서는 ListObjectsV2을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn list_objects(client: &aws_sdk_s3::Client, bucket: &str) -> Result<(),
    S3ExampleError> {
    let mut response = client
        .list_objects_v2()
```

```

        .bucket(bucket.to_owned())
        .max_keys(10) // In this example, go 10 at a time.
        .into_paginator()
        .send();

while let Some(result) = response.next().await {
    match result {
        Ok(output) => {
            for object in output.contents() {
                println!(" - {}", object.key().unwrap_or("Unknown"));
            }
        }
        Err(err) => {
            eprintln!("{err:?}")
        }
    }
}

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListObjectsV2](#)을 참조하십시오.

## PutObject

다음 코드 예시에서는 PutObject을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

pub async fn upload_object(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    file_name: &str,
    key: &str,

```

```

) -> Result<aws_sdk_s3::operation::put_object::PutObjectOutput, S3ExampleError> {
    let body =
aws_sdk_s3::primitives::ByteStream::from_path(std::path::Path::new(file_name)).await;
    client
        .put_object()
        .bucket(bucket_name)
        .key(key)
        .body(body.unwrap())
        .send()
        .await
        .map_err(S3ExampleError::from)
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [PutObject](#)를 참조하십시오.

## UploadPart

다음 코드 예시에서는 UploadPart을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

let mut upload_parts: Vec<aws_sdk_s3::types::CompletedPart> = Vec::new();

for chunk_index in 0..chunk_count {
    let this_chunk = if chunk_count - 1 == chunk_index {
        size_of_last_chunk
    } else {
        CHUNK_SIZE
    };
    let stream = ByteStream::read_from()
        .path(path)
        .offset(chunk_index * CHUNK_SIZE)
        .length(Length::Exact(this_chunk))
        .build()
}

```

```

        .await
        .unwrap();

    // Chunk index needs to start at 0, but part numbers start at 1.
    let part_number = (chunk_index as i32) + 1;
    let upload_part_res = client
        .upload_part()
        .key(&key)
        .bucket(&bucket_name)
        .upload_id(upload_id)
        .body(stream)
        .part_number(part_number)
        .send()
        .await?;

    upload_parts.push(
        CompletedPart::builder()
            .e_tag(upload_part_res.e_tag.unwrap_or_default())
            .part_number(part_number)
            .build(),
    );
}

```

```

// Create a multipart upload. Use UploadPart and CompleteMultipartUpload to
// upload the file.
let multipart_upload_res: CreateMultipartUploadOutput = client
    .create_multipart_upload()
    .bucket(&bucket_name)
    .key(&key)
    .send()
    .await?;

let upload_id = multipart_upload_res.upload_id().ok_or(S3ExampleError::new(
    "Missing upload_id after CreateMultipartUpload",
))?;

```

```

// upload_parts: Vec<aws_sdk_s3::types::CompletedPart>
let completed_multipart_upload: CompletedMultipartUpload =
CompletedMultipartUpload::builder()
    .set_parts(Some(upload_parts))
    .build();

```

```
let _complete_multipart_upload_res = client
    .complete_multipart_upload()
    .bucket(&bucket_name)
    .key(&key)
    .multipart_upload(completed_multipart_upload)
    .upload_id(upload_id)
    .send()
    .await?;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [UploadPart](#)을 참조하십시오.

## 시나리오

텍스트를 스피치로, 다시 스피치에서 텍스트로 변환

다음 코드 예시는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon Polly를 사용하여 일반 텍스트(UTF-8) 입력 파일을 오디오 파일에 합성합니다.
- Amazon S3 버킷에 오디오 파일을 업로드합니다.
- Amazon Transcribe를 사용하여 오디오 파일을 텍스트로 변환합니다.
- 텍스트를 표시합니다.

### SDK for Rust

Amazon Polly를 사용하여 일반 텍스트(UTF-8) 입력 파일을 오디오 파일에 합성하고, 오디오 파일을 Amazon S3 버킷에 업로드하고, Amazon Transcribe를 사용하여 해당 오디오 파일을 텍스트로 변환하고, 텍스트를 표시합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예시를 참조하세요.

이 예시에서 사용되는 서비스

- Amazon Polly
- Amazon S3
- Amazon Transcribe



## 미리 서명된 URL 생성

다음 코드 예제에서는 Amazon S3에 대해 미리 서명된 URL을 생성하고 객체를 업로드하는 방법을 보여줍니다.

### SDK for Rust

#### Note

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

GET S3 객체에 대한 사전 지정 요청을 만듭니다.

```
/// Generate a URL for a presigned GET request.
async fn get_object(
    client: &Client,
    bucket: &str,
    object: &str,
    expires_in: u64,
) -> Result<(), Box<dyn Error>> {
    let expires_in = Duration::from_secs(expires_in);
    let presigned_request = client
        .get_object()
        .bucket(bucket)
        .key(object)
        .presigned(PresigningConfig::expires_in(expires_in)?)
        .await?;

    println!("Object URI: {}", presigned_request.uri());
    let valid_until = chrono::offset::Local::now() + expires_in;
    println!("Valid until: {valid_until}");

    Ok(())
}
```

PUT S3 객체에 대한 사전 지정 요청을 만듭니다.

```
async fn put_object(
    client: &Client,
```

```

    bucket: &str,
    object: &str,
    expires_in: u64,
) -> Result<String, S3ExampleError> {
    let expires_in: std::time::Duration =
std::time::Duration::from_secs(expires_in);
    let expires_in: aws_sdk_s3::presigning::PresigningConfig =
    PresigningConfig::expires_in(expires_in).map_err(|err| {
        S3ExampleError::new(format!(
            "Failed to convert expiration to PresigningConfig: {err:?}")
        ))
    })?;
    let presigned_request = client
        .put_object()
        .bucket(bucket)
        .key(object)
        .presigned(expires_in)
        .await?;

    Ok(presigned_request.uri().into())
}

```

## 사진을 관리하기 위한 서버리스 애플리케이션 만들기

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

### SDK for Rust

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 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

## 이미지에서 얼굴 감지

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon S3 버킷에 이미지를 저장합니다.
- Amazon Rekognition을 사용하여 연령대, 성별, 감정(예제: 웃음) 등의 얼굴 세부 정보를 감지합니다.
- 이러한 세부 정보를 표시합니다.

## SDK for Rust

uploads 접두사를 사용하여 Amazon S3 버킷에 이미지를 저장하고, Amazon Rekognition을 사용하여 연령대, 성별, 감정(예제: 웃음) 등의 얼굴 세부 정보를 감지한 후 이러한 세부 정보를 표시합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하십시오.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3

## 버킷에서 수정된 객체 가져오기

다음 코드 예제에서는 S3 버킷이 마지막 검색 시간 이후 수정되지 않았을 때에 한하여 해당 버킷 내의 객체에서 데이터를 읽는 방법을 보여줍니다.

## SDK for Rust

### Note

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

```
use aws_sdk_s3:::
```

```

    error::SdkError,
    primitives::{ByteStream, DateTime, DateTimeFormat},
    Client,
};
use s3_code_examples::error::S3ExampleError;
use tracing::{error, warn};

const KEY: &str = "key";
const BODY: &str = "Hello, world!";

/// Demonstrate how `if-modified-since` reports that matching objects haven't
/// changed.
///
/// # Steps
/// - Create a bucket.
/// - Put an object in the bucket.
/// - Get the bucket headers.
/// - Get the bucket headers again but only if modified.
/// - Delete the bucket.
#[tokio::main]
async fn main() -> Result<(), S3ExampleError> {
    tracing_subscriber::fmt::init();

    // Get a new UUID to use when creating a unique bucket name.
    let uuid = uuid::Uuid::new_v4();

    // Load the AWS configuration from the environment.
    let client = Client::new(&aws_config::load_from_env().await);

    // Generate a unique bucket name using the previously generated UUID.
    // Then create a new bucket with that name.
    let bucket_name = format!("if-modified-since-{{uuid}}");
    client
        .create_bucket()
        .bucket(bucket_name.clone())
        .send()
        .await?;

    // Create a new object in the bucket whose name is `KEY` and whose
    // contents are `BODY`.
    let put_object_output = client
        .put_object()
        .bucket(bucket_name.as_str())
        .key(KEY)

```

```

        .body(ByteStream::from_static(BODY.as_bytes()))
        .send()
        .await;

// If the `PutObject` succeeded, get the eTag string from it. Otherwise,
// report an error and return an empty string.
let e_tag_1 = match put_object_output {
    Ok(put_object) => put_object.e_tag.unwrap(),
    Err(err) => {
        error!("{err:?}");
        String::new()
    }
};

// Request the object's headers.
let head_object_output = client
    .head_object()
    .bucket(bucket_name.as_str())
    .key(KEY)
    .send()
    .await;

// If the `HeadObject` request succeeded, create a tuple containing the
// values of the headers `last-modified` and `etag`. If the request
// failed, return the error in a tuple instead.
let (last_modified, e_tag_2) = match head_object_output {
    Ok(head_object) => (
        Ok(head_object.last_modified().cloned().unwrap()),
        head_object.e_tag.unwrap(),
    ),
    Err(err) => (Err(err), String::new()),
};

warn!("last modified: {last_modified:?}");
assert_eq!(
    e_tag_1, e_tag_2,
    "PutObject and first GetObject had differing eTags"
);

println!("First value of last_modified: {last_modified:?}");
println!("First tag: {}\n", e_tag_1);

// Send a second `HeadObject` request. This time, the `if_modified_since`
// option is specified, giving the `last_modified` value returned by the

```

```
// first call to `HeadObject`.
//
// Since the object hasn't been changed, and there are no other objects in
// the bucket, there should be no matching objects.

let head_object_output = client
    .head_object()
    .bucket(bucket_name.as_str())
    .key(KEY)
    .if_modified_since(last_modified.unwrap())
    .send()
    .await;

// If the `HeadObject` request succeeded, the result is a tuple containing
// the `last_modified` and `e_tag_1` properties. This is _not_ the expected
// result.
//
// The _expected_ result of the second call to `HeadObject` is an
// `SdkError::ServiceError` containing the HTTP error response. If that's
// the case and the HTTP status is 304 (not modified), the output is a
// tuple containing the values of the HTTP `last-modified` and `etag`
// headers.
//
// If any other HTTP error occurred, the error is returned as an
// `SdkError::ServiceError`.

let (last_modified, e_tag_2) = match head_object_output {
    Ok(head_object) => (
        Ok(head_object.last_modified().cloned().unwrap()),
        head_object.e_tag.unwrap(),
    ),
    Err(err) => match err {
        SdkError::ServiceError(err) => {
            // Get the raw HTTP response. If its status is 304, the
            // object has not changed. This is the expected code path.
            let http = err.raw();
            match http.status().as_u16() {
                // If the HTTP status is 304: Not Modified, return a
                // tuple containing the values of the HTTP
                // `last-modified` and `etag` headers.
                304 => (
                    Ok(DateTime::from_str(
                        http.headers().get("last-modified").unwrap(),
                        DateTimeFormat::HttpDate,
```

```

        )
        .unwrap()),
        http.headers().get("etag").map(|t| t.into()).unwrap(),
    ),
    // Any other HTTP status code is returned as an
    // `SdkError::ServiceError`.
    _ => (Err(SdkError::ServiceError(err)), String::new()),
}
}
// Any other kind of error is returned in a tuple containing the
// error and an empty string.
_ => (Err(err), String::new()),
},
};

warn!("last modified: {last_modified:?}");
assert_eq!(
    e_tag_1, e_tag_2,
    "PutObject and second HeadObject had different eTags"
);

println!("Second value of last modified: {last_modified:?}");
println!("Second tag: {}", e_tag_2);

// Clean up by deleting the object and the bucket.
client
    .delete_object()
    .bucket(bucket_name.as_str())
    .key(KEY)
    .send()
    .await?;

client
    .delete_bucket()
    .bucket(bucket_name.as_str())
    .send()
    .await?;

Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetObject](#)을 참조하십시오.

## EXIF 및 기타 이미지 정보 저장

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져옵니다.
- Amazon S3 버킷에 이미지 파일을 업로드합니다.
- Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(레이블)을 파악합니다.
- EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

### SDK for Rust

JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져오고, 이미지 파일을 Amazon S3 버킷에 업로드 하며, Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(Amazon Rekognition의 레이블)을 파악한 후 EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon Rekognition
- Amazon S3

### SDK를 사용한 단위 및 통합 테스트

다음 코드 예제는 AWS SDK를 사용하여 단위 및 통합 테스트를 작성할 때 모범 사례 기법의 예제를 보여 줍니다.

### SDK for Rust

#### Note

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

테스트 예제용 Cargo.toml



```

[package]
name = "testing-examples"
version = "0.1.0"
authors = [
  "John Disanti <jdisanti@amazon.com>",
  "Doug Schwartz <dougsch@amazon.com>",
]
edition = "2021"

[dependencies]
async-trait = "0.1.51"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-credential-types = { version = "1.0.1", features = [ "hardcoded-credentials", ] }
aws-sdk-s3 = { version = "1.4.0" }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime = { version = "1.0.1", features = ["test-util"] }
aws-smithy-runtime-api = { version = "1.0.1", features = ["test-util"] }
aws-types = { version = "1.0.1" }
clap = { version = "4.4", features = ["derive"] }
http = "0.2.9"
mockall = "0.11.4"
serde_json = "1"
tokio = { version = "1.20.1", features = ["full"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }

[[bin]]
name = "main"
path = "src/main.rs"

```

## automock 및 서비스 래퍼를 사용한 유닛 테스트 예제

```

use aws_sdk_s3 as s3;
#[allow(unused_imports)]
use mockall::automock;

use s3::operation::list_objects_v2::{ListObjectsV2Error, ListObjectsV2Output};

#[cfg(test)]
pub use MockS3Impl as S3;
#[cfg(not(test))]

```

```

pub use S3Impl as S3;

#[allow(dead_code)]
pub struct S3Impl {
    inner: s3::Client,
}

#[cfg_attr(test, automock)]
impl S3Impl {
    #[allow(dead_code)]
    pub fn new(inner: s3::Client) -> Self {
        Self { inner }
    }

    #[allow(dead_code)]
    pub async fn list_objects(
        &self,
        bucket: &str,
        prefix: &str,
        continuation_token: Option<String>,
    ) -> Result<ListObjectsV2Output, s3::error::SdkError<ListObjectsV2Error>> {
        self.inner
            .list_objects_v2()
            .bucket(bucket)
            .prefix(prefix)
            .set_continuation_token(continuation_token)
            .send()
            .await
    }
}

#[allow(dead_code)]
pub async fn determine_prefix_file_size(
    // Now we take a reference to our trait object instead of the S3 client
    // s3_list: ListObjectsService,
    s3_list: S3,
    bucket: &str,
    prefix: &str,
) -> Result<usize, s3::Error> {
    let mut next_token: Option<String> = None;
    let mut total_size_bytes = 0;
    loop {
        let result = s3_list
            .list_objects(bucket, prefix, next_token.take())
    }
}

```

```

        .await?;

    // Add up the file sizes we got back
    for object in result.contents() {
        total_size_bytes += object.size().unwrap_or(0) as usize;
    }

    // Handle pagination, and break the loop if there are no more pages
    next_token = result.next_continuation_token.clone();
    if next_token.is_none() {
        break;
    }
}
Ok(total_size_bytes)
}

#[cfg(test)]
mod test {
    use super::*;
    use mockall::predicate::eq;

    #[tokio::test]
    async fn test_single_page() {
        let mut mock = MockS3Impl::default();
        mock.expect_list_objects()
            .with(eq("test-bucket"), eq("test-prefix"), eq(None))
            .return_once(|_, _, _| {
                Ok(ListObjectsV2Output::builder()
                    .set_contents(Some(vec![
                        // Mock content for ListObjectsV2 response
                        s3::types::Object::builder().size(5).build(),
                        s3::types::Object::builder().size(2).build(),
                    ]))
                    .build())
            });

        // Run the code we want to test with it
        let size = determine_prefix_file_size(mock, "test-bucket", "test-prefix")
            .await
            .unwrap();

        // Verify we got the correct total size back
        assert_eq!(7, size);
    }
}

```

```
#[tokio::test]
async fn test_multiple_pages() {
    // Create the Mock instance with two pages of objects now
    let mut mock = MockS3Impl::default();
    mock.expect_list_objects()
        .with(eq("test-bucket"), eq("test-prefix"), eq(None))
        .return_once(|_, _, _| {
            Ok(ListObjectsV2Output::builder()
                .set_contents(Some(vec![
                    // Mock content for ListObjectsV2 response
                    s3::types::Object::builder().size(5).build(),
                    s3::types::Object::builder().size(2).build(),
                ]))
                .set_next_continuation_token(Some("next".to_string()))
                .build())
        });
    mock.expect_list_objects()
        .with(
            eq("test-bucket"),
            eq("test-prefix"),
            eq(Some("next".to_string())),
        )
        .return_once(|_, _, _| {
            Ok(ListObjectsV2Output::builder()
                .set_contents(Some(vec![
                    // Mock content for ListObjectsV2 response
                    s3::types::Object::builder().size(3).build(),
                    s3::types::Object::builder().size(9).build(),
                ]))
                .build())
        });

    // Run the code we want to test with it
    let size = determine_prefix_file_size(mock, "test-bucket", "test-prefix")
        .await
        .unwrap();

    assert_eq!(19, size);
}
}
```

## StatiReplayClient를 사용한 통합 테스트 예제

```

use aws_sdk_s3 as s3;

#[allow(dead_code)]
pub async fn determine_prefix_file_size(
    // Now we take a reference to our trait object instead of the S3 client
    // s3_list: ListObjectsService,
    s3: s3::Client,
    bucket: &str,
    prefix: &str,
) -> Result<usize, s3::Error> {
    let mut next_token: Option<String> = None;
    let mut total_size_bytes = 0;
    loop {
        let result = s3
            .list_objects_v2()
            .prefix(prefix)
            .bucket(bucket)
            .set_continuation_token(next_token.take())
            .send()
            .await?;

        // Add up the file sizes we got back
        for object in result.contents() {
            total_size_bytes += object.size().unwrap_or(0) as usize;
        }

        // Handle pagination, and break the loop if there are no more pages
        next_token = result.next_continuation_token.clone();
        if next_token.is_none() {
            break;
        }
    }
    Ok(total_size_bytes)
}

#[allow(dead_code)]
fn make_s3_test_credentials() -> s3::config::Credentials {
    s3::config::Credentials::new(
        "ATESTCLIENT",
        "atestsecretkey",
        Some("atestsessiontoken".to_string()),
    )
}

```

```

        None,
        "",
    )
}

#[cfg(test)]
mod test {
    use super::*;
    use aws_config::BehaviorVersion;
    use aws_sdk_s3 as s3;
    use aws_smithy_runtime::client::http::test_util::{ReplayEvent,
StaticReplayClient};
    use aws_smithy_types::body::SdkBody;

    #[tokio::test]
    async fn test_single_page() {
        let page_1 = ReplayEvent::new(
            http::Request::builder()
                .method("GET")
                .uri("https://test-bucket.s3.us-east-1.amazonaws.com/?list-
type=2&prefix=test-prefix")
                .body(SdkBody::empty())
                .unwrap(),
            http::Response::builder()
                .status(200)
                .body(SdkBody::from(include_str!("./testing/response_1.xml")))
                .unwrap(),
        );
        let replay_client = StaticReplayClient::new(vec![page_1]);
        let client: s3::Client = s3::Client::from_conf(
            s3::Config::builder()
                .behavior_version(BehaviorVersion::latest())
                .credentials_provider(make_s3_test_credentials())
                .region(s3::config::Region::new("us-east-1"))
                .http_client(replay_client.clone())
                .build(),
        );

        // Run the code we want to test with it
        let size = determine_prefix_file_size(client, "test-bucket", "test-prefix")
            .await
            .unwrap();

        // Verify we got the correct total size back

```

```

    assert_eq!(7, size);
    replay_client.assert_requests_match(&[]);
}

#[tokio::test]
async fn test_multiple_pages() {
    let page_1 = ReplayEvent::new(
        http::Request::builder()
            .method("GET")
            .uri("https://test-bucket.s3.us-east-1.amazonaws.com/?list-
type=2&prefix=test-prefix")
            .body(SdkBody::empty())
            .unwrap(),
        http::Response::builder()
            .status(200)
            .body(SdkBody::from(include_str!("./testing/
response_multi_1.xml")))
            .unwrap(),
    );
    let page_2 = ReplayEvent::new(
        http::Request::builder()
            .method("GET")
            .uri("https://test-bucket.s3.us-east-1.amazonaws.com/?list-
type=2&prefix=test-prefix&continuation-token=next")
            .body(SdkBody::empty())
            .unwrap(),
        http::Response::builder()
            .status(200)
            .body(SdkBody::from(include_str!("./testing/
response_multi_2.xml")))
            .unwrap(),
    );
    let replay_client = StaticReplayClient::new(vec![page_1, page_2]);
    let client: s3::Client = s3::Client::from_conf(
        s3::Config::builder()
            .behavior_version(BehaviorVersion::latest())
            .credentials_provider(make_s3_test_credentials())
            .region(s3::config::Region::new("us-east-1"))
            .http_client(replay_client.clone())
            .build(),
    );

    // Run the code we want to test with it
    let size = determine_prefix_file_size(client, "test-bucket", "test-prefix")

```

```

        .await
        .unwrap();

    assert_eq!(19, size);

    replay_client.assert_requests_match(&[]);
}
}

```

## 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 참조하십시오.

## SDK for Rust

### Note

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

```

use std::fs::File;
use std::io::prelude::*;
use std::path::Path;

use aws_config::meta::region::RegionProviderChain;
use aws_sdk_s3::error::DisplayErrorContext;
use aws_sdk_s3::operation::{
    create_multipart_upload::CreateMultipartUploadOutput,
    get_object::GetObjectOutput,
};
use aws_sdk_s3::types::{CompletedMultipartUpload, CompletedPart};
use aws_sdk_s3::{config::Region, Client as S3Client};
use aws_smithy_types::byte_stream::{ByteStream, Length};
use rand::distributions::Alphanumeric;
use rand::{thread_rng, Rng};

```



```

use s3_code_examples::error::S3ExampleError;
use std::process;
use uuid::Uuid;

//In bytes, minimum chunk size of 5MB. Increase CHUNK_SIZE to send larger chunks.
const CHUNK_SIZE: u64 = 1024 * 1024 * 5;
const MAX_CHUNKS: u64 = 10000;

#[tokio::main]
pub async fn main() {
    if let Err(err) = run_example().await {
        eprintln!("Error: {}", DisplayErrorContext(err));
        process::exit(1);
    }
}

async fn run_example() -> Result<(), S3ExampleError> {
    let shared_config = aws_config::load_from_env().await;
    let client = S3Client::new(&shared_config);

    let bucket_name = format!("amzn-s3-demo-bucket-{}", Uuid::new_v4());
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
    let region = region_provider.region().await.unwrap();
    s3_code_examples::create_bucket(&client, &bucket_name, &region).await?;

    let key = "sample.txt".to_string();
    // Create a multipart upload. Use UploadPart and CompleteMultipartUpload to
    // upload the file.
    let multipart_upload_res: CreateMultipartUploadOutput = client
        .create_multipart_upload()
        .bucket(&bucket_name)
        .key(&key)
        .send()
        .await?;

    let upload_id = multipart_upload_res.upload_id().ok_or(S3ExampleError::new(
        "Missing upload_id after CreateMultipartUpload",
    ))?;

    //Create a file of random characters for the upload.
    let mut file = File::create(&key).expect("Could not create sample file.");
    // Loop until the file is 5 chunks.
    while file.metadata().unwrap().len() <= CHUNK_SIZE * 4 {
        let rand_string: String = thread_rng()

```

```
        .sample_iter(&Alphanumeric)
        .take(256)
        .map(char::from)
        .collect();
let return_string: String = "\n".to_string();
file.write_all(rand_string.as_ref())
    .expect("Error writing to file.");
file.write_all(return_string.as_ref())
    .expect("Error writing to file.");
}

let path = Path::new(&key);
let file_size = tokio::fs::metadata(path)
    .await
    .expect("it exists I swear")
    .len();

let mut chunk_count = (file_size / CHUNK_SIZE) + 1;
let mut size_of_last_chunk = file_size % CHUNK_SIZE;
if size_of_last_chunk == 0 {
    size_of_last_chunk = CHUNK_SIZE;
    chunk_count -= 1;
}

if file_size == 0 {
    return Err(S3ExampleError::new("Bad file size."));
}
if chunk_count > MAX_CHUNKS {
    return Err(S3ExampleError::new(
        "Too many chunks! Try increasing your chunk size.",
    ));
}

let mut upload_parts: Vec<aws_sdk_s3::types::CompletedPart> = Vec::new();

for chunk_index in 0..chunk_count {
    let this_chunk = if chunk_count - 1 == chunk_index {
        size_of_last_chunk
    } else {
        CHUNK_SIZE
    };
    let stream = ByteStream::read_from()
        .path(path)
        .offset(chunk_index * CHUNK_SIZE)
```

```
        .length(Length::Exact(this_chunk))
        .build()
        .await
        .unwrap();

// Chunk index needs to start at 0, but part numbers start at 1.
let part_number = (chunk_index as i32) + 1;
let upload_part_res = client
    .upload_part()
    .key(&key)
    .bucket(&bucket_name)
    .upload_id(upload_id)
    .body(stream)
    .part_number(part_number)
    .send()
    .await?;

upload_parts.push(
    CompletedPart::builder()
        .e_tag(upload_part_res.e_tag.unwrap_or_default())
        .part_number(part_number)
        .build(),
);
}

// upload_parts: Vec<aws_sdk_s3::types::CompletedPart>
let completed_multipart_upload: CompletedMultipartUpload =
CompletedMultipartUpload::builder()
    .set_parts(Some(upload_parts))
    .build();

let _complete_multipart_upload_res = client
    .complete_multipart_upload()
    .bucket(&bucket_name)
    .key(&key)
    .multipart_upload(completed_multipart_upload)
    .upload_id(upload_id)
    .send()
    .await?;

let data: GetObjectOutput =
    s3_code_examples::download_object(&client, &bucket_name, &key).await?;
let data_length: u64 = data
    .content_length()
```

```

        .unwrap_or_default()
        .try_into()
        .unwrap();
    if file.metadata().unwrap().len() == data_length {
        println!("Data lengths match.");
    } else {
        println!("The data was not the same size!");
    }

    s3_code_examples::clear_bucket(&client, &bucket_name)
        .await
        .expect("Error emptying bucket.");
    s3_code_examples::delete_bucket(&client, &bucket_name)
        .await
        .expect("Error deleting bucket.");

    Ok(())
}

```

## 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 Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 S3 이벤트를 사용합니다.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;

```

```
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request from
SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket = evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket name
to exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object key to
exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
```

```
        .bucket(bucket)
        .key(key)
        .send()
        .await;

    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}
```

## SDK for Rust를 사용한 SageMaker AI 예제

다음 코드 예제에서는 SageMaker AI와 함께 AWS SDK for Rust를 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

#### ListNotebookInstances

다음 코드 예시에서는 ListNotebookInstances을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```

async fn show_instances(client: &Client) -> Result<(), Error> {
    let notebooks = client.list_notebook_instances().send().await?;

    println!("Notebooks:");

    for n in notebooks.notebook_instances() {
        let n_instance_type = n.instance_type().unwrap();
        let n_status = n.notebook_instance_status().unwrap();
        let n_name = n.notebook_instance_name();

        println!("  Name :          {}", n_name.unwrap_or("Unknown"));
        println!("  Status :         {}", n_status.as_ref());
        println!("  Instance Type : {}", n_instance_type.as_ref());
        println!();
    }

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListNotebookInstances](#)를 참조하십시오.

**ListTrainingJobs**

다음 코드 예시에서는 ListTrainingJobs을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```

async fn show_jobs(client: &Client) -> Result<(), Error> {
    let job_details = client.list_training_jobs().send().await?;

    println!("Jobs:");

    for j in job_details.training_job_summaries() {
        let name = j.training_job_name().unwrap_or("Unknown");
        let creation_time = j.creation_time().expect("creation
time").to_chrono_utc()?;
        let training_end_time = j
            .training_end_time()
            .expect("Training end time")
            .to_chrono_utc()?;

        let status = j.training_job_status().expect("training status");
        let duration = training_end_time - creation_time;

        println!(" Name:                {}", name);
        println!(
            " Creation date/time: {}",
            creation_time.format("%Y-%m-%d@%H:%M:%S")
        );
        println!(" Duration (seconds): {}", duration.num_seconds());
        println!(" Status:                {:?}" , status);

        println!();
    }

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListTrainingJobs](#)을 참조하십시오.

## SDK for Rust를 사용한 Secrets Manager

다음 코드 예제에서는 Secrets Manager와 함께 AWS SDK for Rust를 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 예시에서는 GetSecretValue을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn show_secret(client: &Client, name: &str) -> Result<(), Error> {
    let resp = client.get_secret_value().secret_id(name).send().await?;

    println!("Value: {}", resp.secret_string().unwrap_or("No value!"));

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [GetSecretValue](#)를 참조하십시오.

## SDK for Rust를 사용한 Amazon SES API v2 예제

다음 코드 예제에서는 Amazon SES API v2와 함께 AWS SDK for Rust를 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.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 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

### CreateContact

다음 코드 예시에서는 CreateContact을 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn add_contact(client: &Client, list: &str, email: &str) -> Result<(), Error>
{
    client
        .create_contact()
        .contact_list_name(list)
        .email_address(email)
        .send()
        .await?;

    println!("Created contact");
}
```

```
    Ok(())
}
```

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

## CreateContactList

다음 코드 예시에서는 CreateContactList를 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```
async fn make_list(client: &Client, contact_list: &str) -> Result<(), Error> {
    client
        .create_contact_list()
        .contact_list_name(contact_list)
        .send()
        .await?;

    println!("Created contact list.");

    Ok(())
}
```

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

## CreateEmailIdentity

다음 코드 예시에서는 CreateEmailIdentity를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
match self
  .client
  .create_email_identity()
  .email_identity(self.verified_email.clone())
  .send()
  .await
{
  Ok(_) => writeln!(self.stdout, "Email identity created successfully.")?,
  Err(e) => match e.into_service_error() {
    CreateEmailIdentityError::AlreadyExistsException(_) => {
      writeln!(
        self.stdout,
        "Email identity already exists, skipping creation."
      )?;
    }
  },
  e => return Err( anyhow!("Error creating email identity: {}", e) ),
},
}
```

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

**CreateEmailTemplate**

다음 코드 예시에서는 CreateEmailTemplate을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```

let template_html =
    std::fs::read_to_string("../resources/newsletter/coupon-
newsletter.html")
        .unwrap_or_else(|_| "Missing coupon-newsletter.html".to_string());
let template_text =
    std::fs::read_to_string("../resources/newsletter/coupon-newsletter.txt")
        .unwrap_or_else(|_| "Missing coupon-newsletter.txt".to_string());

// Create the email template
let template_content = EmailTemplateContent::builder()
    .subject("Weekly Coupons Newsletter")
    .html(template_html)
    .text(template_text)
    .build();

match self
    .client
    .create_email_template()
    .template_name(TEMPLATE_NAME)
    .template_content(template_content)
    .send()
    .await
{
    Ok(_) => writeln!(self.stdout, "Email template created successfully.")?,
    Err(e) => match e.into_service_error() {
        CreateEmailTemplateError::AlreadyExistsException(_) => {
            writeln!(
                self.stdout,
                "Email template already exists, skipping creation."
            )?;
        }
        e => return Err( anyhow!("Error creating email template: {}", e)),
    },
}

```

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

## DeleteContactList

다음 코드 예시에서는 DeleteContactList을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
match self
  .client
  .delete_contact_list()
  .contact_list_name(CONTACT_LIST_NAME)
  .send()
  .await
{
  Ok(_) => writeln!(self.stdout, "Contact list deleted successfully.")?,
  Err(e) => return Err( anyhow!("Error deleting contact list: {e}") ),
}
```

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

**DeleteEmailIdentity**

다음 코드 예시에서는 DeleteEmailIdentity을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
match self
  .client
  .delete_email_identity()
  .email_identity(self.verified_email.clone())
  .send()
```

```

        .await
    {
        Ok(_) => writeln!(self.stdout, "Email identity deleted
successfully.")?,
        Err(e) => {
            return Err(anyhow!("Error deleting email identity: {}", e));
        }
    }
}

```

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

## DeleteEmailTemplate

다음 코드 예시에서는 DeleteEmailTemplate을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

```

match self
    .client
    .delete_email_template()
    .template_name(TEMPLATE_NAME)
    .send()
    .await
{
    Ok(_) => writeln!(self.stdout, "Email template deleted successfully.")?,
    Err(e) => {
        return Err(anyhow!("Error deleting email template: {e}"));
    }
}

```

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

## GetEmailIdentity

다음 코드 예시에서는 GetEmailIdentity을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

이메일 주소가 확인되었는지 여부를 결정합니다.

```
async fn is_verified(client: &Client, email: &str) -> Result<(), Error> {
    let resp = client
        .get_email_identity()
        .email_identity(email)
        .send()
        .await?;

    if resp.verified_for_sending_status() {
        println!("The address is verified");
    } else {
        println!("The address is not verified");
    }

    Ok(())
}
```

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

## ListContactLists

다음 코드 예시에서는 ListContactLists을 사용하는 방법을 보여 줍니다.



## SDK for Rust

**Note**

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

```
async fn show_lists(client: &Client) -> Result<(), Error> {
    let resp = client.list_contact_lists().send().await?;

    println!("Contact lists:");

    for list in resp.contact_lists() {
        println!("  {}", list.contact_list_name().unwrap_or_default());
    }

    Ok(())
}
```

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

**ListContacts**

다음 코드 예시에서는 ListContacts를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn show_contacts(client: &Client, list: &str) -> Result<(), Error> {
    let resp = client
        .list_contacts()
        .contact_list_name(list)
        .send()
}
```

```

        .await?;

println!("Contacts:");

for contact in resp.contacts() {
    println!(" {}", contact.email_address().unwrap_or_default());
}

Ok(())
}

```

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

## SendEmail

다음 코드 예시에서는 SendEmail을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

연락처 목록의 모든 구성원에게 메시지를 전송합니다.

```

async fn send_message(
    client: &Client,
    list: &str,
    from: &str,
    subject: &str,
    message: &str,
) -> Result<(), Error> {
    // Get list of email addresses from contact list.
    let resp = client
        .list_contacts()
        .contact_list_name(list)
        .send()
        .await?;
}

```

```
let contacts = resp.contacts();

let cs: Vec<String> = contacts
    .iter()
    .map(|i| i.email_address().unwrap_or_default().to_string())
    .collect();

let mut dest: Destination = Destination::builder().build();
dest.to_addresses = Some(cs);
let subject_content = Content::builder()
    .data(subject)
    .charset("UTF-8")
    .build()
    .expect("building Content");
let body_content = Content::builder()
    .data(message)
    .charset("UTF-8")
    .build()
    .expect("building Content");
let body = Body::builder().text(body_content).build();

let msg = Message::builder()
    .subject(subject_content)
    .body(body)
    .build();

let email_content = EmailContent::builder().simple(msg).build();

client
    .send_email()
    .from_email_address(from)
    .destination(dest)
    .content(email_content)
    .send()
    .await?;

println!("Email sent to list");

Ok(())
}
```

연락처 목록의 모든 구성원에게 템플릿을 사용하여 메시지를 전송합니다.

```

    let coupons = std::fs::read_to_string("../resources/newsletter/
sample_coupons.json")
        .unwrap_or_else(|_| r#"{"coupons":[]}"#.to_string());
    let email_content = EmailContent::builder()
        .template(
            Template::builder()
                .template_name(TEMPLATE_NAME)
                .template_data(coupons)
                .build(),
        )
        .build();

    match self
        .client
        .send_email()
        .from_email_address(self.verified_email.clone())

        .destination(Destination::builder().to_addresses(email.clone()).build())
        .content(email_content)
        .list_management_options(
            ListManagementOptions::builder()
                .contact_list_name(CONTACT_LIST_NAME)
                .build()?,
        )
        .send()
        .await
    {
        Ok(output) => {
            if let Some(message_id) = output.message_id {
                writeln!(
                    self.stdout,
                    "Newsletter sent to {} with message ID {}",
                    email, message_id
                )?;
            } else {
                writeln!(self.stdout, "Newsletter sent to {}", email)?;
            }
        }
        Err(e) => return Err( anyhow!("Error sending newsletter to {}: {}",
email, e)),
    }

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [SendEmail](#)을 참조하십시오.

## 시나리오

### 뉴스레터 시나리오

다음 코드 예제는 Amazon SES API v2 뉴스레터 시나리오를 실행하는 방법을 보여줍니다.

### SDK for Rust

#### Note

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

```

match self
    .client
    .create_contact_list()
    .contact_list_name(CONTACT_LIST_NAME)
    .send()
    .await
{
    Ok(_) => writeln!(self.stdout, "Contact list created successfully.")?,
    Err(e) => match e.into_service_error() {
        CreateContactListError::AlreadyExistsException(_) => {
            writeln!(
                self.stdout,
                "Contact list already exists, skipping creation."
            )?;
        }
        e => return Err(anyhow!("Error creating contact list: {}", e)),
    },
}

match self
    .client
    .create_contact()
    .contact_list_name(CONTACT_LIST_NAME)
    .email_address(email.clone())
    .send()
    .await

```

```

    {
        Ok(_) => writeln!(self.stdout, "Contact created for {}", email)?,
        Err(e) => match e.into_service_error() {
            CreateContactError::AlreadyExistsException(_) => writeln!(
                self.stdout,
                "Contact already exists for {}, skipping creation.",
                email
            )?,
            e => return Err(anyhow!("Error creating contact for {}: {}",
email, e)),
        },
    }

    let contacts: Vec<Contact> = match self
        .client
        .list_contacts()
        .contact_list_name(CONTACT_LIST_NAME)
        .send()
        .await
    {
        Ok(list_contacts_output) => {
            list_contacts_output.contacts.unwrap().into_iter().collect()
        }
        Err(e) => {
            return Err(anyhow!(
                "Error retrieving contact list {}: {}",
                CONTACT_LIST_NAME,
                e
            ))
        }
    };

    let coupons = std::fs::read_to_string("../resources/newsletter/
sample_coupons.json")
        .unwrap_or_else(|_| r#"{"coupons":[]}"#.to_string());
    let email_content = EmailContent::builder()
        .template(
            Template::builder()
                .template_name(TEMPLATE_NAME)
                .template_data(coupons)
                .build(),
        )
        .build();

```

```

        match self
            .client
            .send_email()
            .from_email_address(self.verified_email.clone())

        .destination(Destination::builder().to_addresses(email.clone()).build())
        .content(email_content)
        .list_management_options(
            ListManagementOptions::builder()
                .contact_list_name(CONTACT_LIST_NAME)
                .build()?,
        )
        .send()
        .await
    {
        Ok(output) => {
            if let Some(message_id) = output.message_id {
                writeln!(
                    self.stdout,
                    "Newsletter sent to {} with message ID {}",
                    email, message_id
                )?;
            } else {
                writeln!(self.stdout, "Newsletter sent to {}", email)?;
            }
        }
        Err(e) => return Err( anyhow!("Error sending newsletter to {}: {}",
email, e)),
    }

    match self
        .client
        .create_email_identity()
        .email_identity(self.verified_email.clone())
        .send()
        .await
    {
        Ok(_) => writeln!(self.stdout, "Email identity created successfully.")?,
        Err(e) => match e.into_service_error() {
            CreateEmailIdentityError::AlreadyExistsException(_) => {
                writeln!(
                    self.stdout,
                    "Email identity already exists, skipping creation."
                )?;
            }
        }
    }

```

```

        }
        e => return Err(anyhow!("Error creating email identity: {}", e)),
    },
}

let template_html =
    std::fs::read_to_string("../resources/newsletter/coupon-
newsletter.html")
        .unwrap_or_else(|_| "Missing coupon-newsletter.html".to_string());
let template_text =
    std::fs::read_to_string("../resources/newsletter/coupon-newsletter.txt")
        .unwrap_or_else(|_| "Missing coupon-newsletter.txt".to_string());

// Create the email template
let template_content = EmailTemplateContent::builder()
    .subject("Weekly Coupons Newsletter")
    .html(template_html)
    .text(template_text)
    .build();

match self
    .client
    .create_email_template()
    .template_name(TEMPLATE_NAME)
    .template_content(template_content)
    .send()
    .await
{
    Ok(_) => writeln!(self.stdout, "Email template created successfully.")?,
    Err(e) => match e.into_service_error() {
        CreateEmailTemplateError::AlreadyExistsException(_) => {
            writeln!(
                self.stdout,
                "Email template already exists, skipping creation."
            )?;
        }
        e => return Err(anyhow!("Error creating email template: {}", e)),
    },
}

match self
    .client
    .delete_contact_list()
    .contact_list_name(CONTACT_LIST_NAME)

```



```

        .send()
        .await
    {
        Ok(_) => writeln!(self.stdout, "Contact list deleted successfully.")?,
        Err(e) => return Err(anyhow!("Error deleting contact list: {e}")),
    }

    match self
        .client
        .delete_email_identity()
        .email_identity(self.verified_email.clone())
        .send()
        .await
    {
        Ok(_) => writeln!(self.stdout, "Email identity deleted
successfully.")?,
        Err(e) => {
            return Err(anyhow!("Error deleting email identity: {}", e));
        }
    }

    match self
        .client
        .delete_email_template()
        .template_name(TEMPLATE_NAME)
        .send()
        .await
    {
        Ok(_) => writeln!(self.stdout, "Email template deleted successfully.")?,
        Err(e) => {
            return Err(anyhow!("Error deleting email template: {e}"));
        }
    }
}

```

• API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하세요.

- [CreateContact](#)
- [CreateContactList](#)
- [CreateEmailIdentity](#)
- [CreateEmailTemplate](#)
- [DeleteContactList](#)

- [DeleteEmailIdentity](#)
- [DeleteEmailTemplate](#)
- [ListContacts](#)
- [SendEmail.simple](#)
- [SendEmail.template](#)

## SDK for Rust를 사용한 Amazon SNS 예제

다음 코드 예제에서는 Amazon SNS에서 AWS SDK for Rust를 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.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 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](#)
- [서버리스 예제](#)

작업

### CreateTopic

다음 코드 예시에서는 CreateTopic을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn make_topic(client: &Client, topic_name: &str) -> Result<(), Error> {
    let resp = client.create_topic().name(topic_name).send().await?;

    println!(
        "Created topic with ARN: {}",
        resp.topic_arn().unwrap_or_default()
    );

    Ok(())
}
```

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

**ListTopics**

다음 코드 예시에서는 ListTopics를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn show_topics(client: &Client) -> Result<(), Error> {
    let resp = client.list_topics().send().await?;

    println!("Topic ARNs:");

    for topic in resp.topics() {
```

```

        println!("{}", topic.topic_arn().unwrap_or_default());
    }

    Ok(())
}

```

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

## Publish

다음 코드 예시에서는 Publish을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`", topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")

```

```

        .send()
        .await?;

println!("Published message: {:?}", rsp);

Ok(())
}

```

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

## Subscribe

다음 코드 예시에서는 Subscribe을 사용하는 방법을 보여 줍니다.

SDK for Rust

### Note

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

이메일 주소로 주제 구독.

```

async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`", topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);
}

```

```
    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;

    println!("Published message: {:?}", rsp);

    Ok(())
}
```

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

## 시나리오

### 사진을 관리하기 위한 서버리스 애플리케이션 만들기

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

#### SDK for Rust

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 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- 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 Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 SNS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
// aws_lambda_events = { version = "0.10.0", default-features = false, features = ["sns"] }
// lambda_runtime = "0.8.1"
// tokio = { version = "1", features = ["macros"] }
// tracing = { version = "0.1", features = ["log"] }
// tracing-subscriber = { version = "0.3", default-features = false, features = ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}
```

```
fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);

    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

## SDK for Rust를 사용한 Amazon SQS 예제

다음 코드 예제에서는 Amazon SQS에서 AWS SDK for Rust를 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

#### ListQueues

다음 코드 예시에서는 ListQueues을 사용하는 방법을 보여 줍니다.



## SDK for Rust

**Note**

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

지역에 나열된 첫 번째 Amazon SQS 대기열을 검색합니다.

```
async fn find_first_queue(client: &Client) -> Result<String, Error> {
    let queues = client.list_queues().send().await?;
    let queue_urls = queues.queue_urls();
    Ok(queue_urls
        .first()
        .expect("No queues in this account and Region. Create a queue to proceed.")
        .to_string())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ListQueues](#)를 참조하십시오.

**ReceiveMessage**

다음 코드 예시에서는 ReceiveMessage을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
async fn receive(client: &Client, queue_url: &String) -> Result<(), Error> {
    let rcv_message_output =
        client.receive_message().queue_url(queue_url).send().await?;

    println!("Messages from queue with url: {}", queue_url);
}
```

```

    for message in rcv_message_output.messages.unwrap_or_default() {
        println!("Got the message: {:#?}", message);
    }

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [ReceiveMessage](#)를 참조하십시오.

## SendMessage

다음 코드 예시에서는 SendMessage을 사용하는 방법을 보여 줍니다.

### SDK for Rust

#### Note

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

```

async fn send(client: &Client, queue_url: &String, message: &SQSMessage) ->
    Result<(), Error> {
    println!("Sending message to queue with URL: {}", queue_url);

    let rsp = client
        .send_message()
        .queue_url(queue_url)
        .message_body(&message.body)
        // If the queue is FIFO, you need to set .message_deduplication_id
        // and message_group_id or configure the queue for
        ContentBasedDeduplication.
        .send()
        .await?;

    println!("Send message to the queue: {:#?}", rsp);

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [SendMessage](#)를 참조하십시오.

## 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 Rust

### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 SQS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
            record.body.as_deref().unwrap_or_default());
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
```

```

        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## 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 Rust

#### Note

GitHub에 더 많은 내용이 있습니다. [서버리스 예제](#) 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 SQS 배치 항목 실패를 보고합니다.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<SqsBatchResponse,
Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {

```

```

        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}

```

## AWS STS SDK for Rust를 사용한 예제

다음 코드 예제에서는 AWS SDK for Rust를 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 예시에서는 AssumeRole을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```

async fn assume_role(config: &SdkConfig, role_name: String, session_name:
Option<String>) {
    let provider = aws_config::sts::AssumeRoleProvider::builder(role_name)
        .session_name(session_name.unwrap_or("rust_sdk_example_session".into()))
        .configure(config)
        .build()
        .await;

    let local_config = aws_config::from_env()
        .credentials_provider(provider)
        .load()
        .await;

    let client = Client::new(&local_config);
    let req = client.get_caller_identity();
    let resp = req.send().await;
    match resp {
        Ok(e) => {
            println!("UserID :           {}", e.user_id().unwrap_or_default());
            println!("Account:           {}", e.account().unwrap_or_default());
            println!("Arn      :           {}", e.arn().unwrap_or_default());
        }
        Err(e) => println!("{:?}", e),
    }
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [AssumeRole](#)을 참조하십시오.

## SDK for Rust를 사용한 Systems Manager 예제

다음 코드 예제에서는 Systems Manager에서 AWS SDK for Rust를 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

### DescribeParameters

다음 코드 예시에서는 DescribeParameters를 사용하는 방법을 보여 줍니다.

SDK for Rust

#### Note

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

```
async fn show_parameters(client: &Client) -> Result<(), Error> {
    let resp = client.describe_parameters().send().await?;

    for param in resp.parameters() {
        println!("{}", param.name().unwrap_or_default());
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [DescribeParameters](#)를 참조하십시오.

### GetParameter

다음 코드 예시에서는 GetParameter를 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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

```
pub async fn list_path(&self, path: &str) -> Result<Vec<Parameter>, EC2Error> {
    let maybe_params: Vec<Result<Parameter, _>> = TryFlatMap::new(
        self.inner
            .get_parameters_by_path()
            .path(path)
            .into_paginator()
            .send(),
    )
    .flat_map(|item| item.parameters.unwrap_or_default())
    .collect()
    .await;
    // Fail on the first error
    let params = maybe_params
        .into_iter()
        .collect:::<Result<Vec<Parameter>, _>>()?;
    Ok(params)
}
```

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

**PutParameter**

다음 코드 예시에서는 PutParameter을 사용하는 방법을 보여 줍니다.

## SDK for Rust

**Note**

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



```

async fn make_parameter(
    client: &Client,
    name: &str,
    value: &str,
    description: &str,
) -> Result<(), Error> {
    let resp = client
        .put_parameter()
        .overwrite(true)
        .r#type(ParameterType::String)
        .name(name)
        .value(value)
        .description(description)
        .send()
        .await?;

    println!("Success! Parameter now has version: {}", resp.version());

    Ok(())
}

```

- API 세부 정보는 AWS SDK for Rust API 참조의 [PutParameter](#)를 참조하십시오.

## SDK for Rust를 사용한 Amazon Transcribe 예제

다음 코드 예제에서는 Amazon Transcribe에서 AWS SDK for Rust를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 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

텍스트를 스피치로, 다시 스피치에서 텍스트로 변환

다음 코드 예시는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon Polly를 사용하여 일반 텍스트(UTF-8) 입력 파일을 오디오 파일에 합성합니다.
- Amazon S3 버킷에 오디오 파일을 업로드합니다.
- Amazon Transcribe를 사용하여 오디오 파일을 텍스트로 변환합니다.
- 텍스트를 표시합니다.

### SDK for Rust

Amazon Polly를 사용하여 일반 텍스트(UTF-8) 입력 파일을 오디오 파일에 합성하고, 오디오 파일을 Amazon S3 버킷에 업로드하고, Amazon Transcribe를 사용하여 해당 오디오 파일을 텍스트로 변환하고, 텍스트를 표시합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 [GitHub](#)에서 전체 예시를 참조하세요.

이 예시에서 사용되는 서비스

- Amazon Polly
- Amazon S3
- Amazon Transcribe

## 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. 보안 책임은에서 가장 중요하며 AWS, 보안의 효율성은 [AWS 규정 준수 프로그램의](#) 일환으로 타사 감사자가 정기적으로 테스트하고 확인합니다.

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 모델을](#) 따릅니다. AWS 서비스 보안 정보는 [AWS 서비스 보안 설명서 페이지](#) 및 규정 [AWSAWS 준수 프로그램의 규정 준수 노력 범위에 속하는 서비스를 참조하세요.](#)

### 주제

- [이 AWS 제품 또는 서비스의 데이터 보호](#)
- [이 AWS 제품 또는 서비스에 대한 규정 준수 검증](#)
- [이 AWS 제품 또는 서비스에 대한 인프라 보안](#)
- [에서 최소 TLS 버전 적용 AWS SDK for Rust](#)

## 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 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 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 사

용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- 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.
- AWS 암호화 솔루션과 내부의 모든 기본 보안 제어를 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 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 [Federal Information Processing Standard\(FIPS\) 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 서비스를 사용하거나 콘솔, API 또는 AWS CLI AWS SDKs를 사용하는 다른 AWS 서비스 가 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 자격 증명을 URL에 포함해서는 안 됩니다.

## 이 AWS 제품 또는 서비스에 대한 규정 준수 검증

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 규정 준수 [AWS 서비스 프로그램 범위 규정 준수](#) 섹션을 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 in Downloading AWS Artifact](#) 참조하세요.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 AWS 서비스 결정됩니다.는 규정 준수를 지원하기 위해 다음 리소스를 AWS 제공합니다.

- [보안 규정 준수 및 거버넌스](#) - 이러한 솔루션 구현 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 기능을 배포하는 단계를 제공합니다.
- [HIPAA 적격 서비스 참조](#)-HIPAA 적격 서비스가 나열되어 있습니다. 모두가 HIPAA에 적합한 AWS 서비스 것은 아닙니다.

- [AWS 규정 준수 리소스](#) - 이 워크북 및 가이드 모음은 업계 및 위치에 적용될 수 있습니다.
- [AWS 고객 규정 준수 가이드](#) - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에는 여러 프레임워크(미국 국립표준기술연구소(NIST), 결제카드 산업 보안 표준 위원회(PCI), 국제표준화기구(ISO))의 보안 제어에 대한 지침을 보호하고 AWS 서비스 매핑하는 모범 사례가 요약되어 있습니다.
- AWS Config 개발자 안내서의 [규칙을 사용하여 리소스 평가](#) - 이 AWS Config 서비스는 리소스 구성 이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- [AWS Security Hub](#) - 이를 AWS 서비스 통해 내 보안 상태를 포괄적으로 확인할 수 있습니다 AWS. Security Hub는 보안 컨트롤을 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 [Security Hub 제어 참조](#)를 참조하세요.
- [Amazon GuardDuty](#) - 의심스러운 악의적인 활동이 있는지 환경을 모니터링하여 사용자, AWS 계정 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty는 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 따르는 데 도움을 줄 수 있습니다.
- [AWS Audit Manager](#) - 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 규정 및 업계 표준의 위험 및 규정 준수를 관리하는 방법을 간소화할 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 노력 범위에 속하는 서비스를 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 보안을](#) 참조하세요. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 Security Pillar AWS Well-Architected Framework의 [인프라 보호](#)를 참조하세요.

AWS 게시된 API 호출을 사용하여 네트워크를 통해이 AWS 제품 또는 서비스에 액세스합니다. 고객은 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 보안 암호 액세스 키를 사용하여 서명해야 합니다. 또는 [AWS Security Token Service](#)(AWS STS)를 사용하여 임시 자격 증명을 생성하여 요청에 서명할 수 있습니다.

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 노력 범위에 속하는 서비스를 참조하세요](#).

## 에서 최소 TLS 버전 적용 AWS SDK for Rust

는 AWS 서비스와 통신할 때 TLS를 AWS SDK for Rust 사용하여 보안을 강화합니다. SDK는 기본적으로 최소 TLS 버전 1.2를 적용합니다. 기본적으로 SDK는 클라이언트 애플리케이션과 서비스 모두에서 사용할 수 있는 최고 버전의 TLS도 협상합니다. 예를 들어 SDK는 TLS 1.3을 협상할 수 있습니다.

SDK가 사용하는 TCP 커넥터의 수동 구성을 제공하여 애플리케이션에서 특정 TLS 버전을 적용할 수 있습니다. 이를 설명하기 위해 다음 예제에서는 TLS 1.3을 적용하는 방법을 보여줍니다.

### Note

일부 AWS 서비스는 아직 TLS 1.3을 지원하지 않으므로 이 버전을 적용하면 SDK 상호 운용성에 영향을 미칠 수 있습니다. 프로덕션 배포 전에 각 서비스로 이 구성을 테스트하는 것이 좋습니다.

```
pub async fn connect_via_tls_13() -> Result<(), Error> {
    println!("Attempting to connect to KMS using TLS 1.3: ");

    // Let webpki load the Mozilla root certificates.
    let mut root_store = RootCertStore::empty();
    root_store.add_server_trust_anchors(webpki_roots::TLS_SERVER_ROOTS.0.iter().map(|
ta| {
        rustls::OwnedTrustAnchor::from_subject_spki_name_constraints(
            ta.subject,
            ta.spki,
            ta.name_constraints,
        )
    }));

    // The .with_protocol_versions call is where we set TLS1.3. You can add
    rustls::version::TLS12 or replace them both with rustls::ALL_VERSIONS
```

```
let config = rustls::ClientConfig::builder()
    .with_safe_default_cipher_suites()
    .with_safe_default_kx_groups()
    .with_protocol_versions(&[&rustls::version::TLS13])
    .expect("It looks like your system doesn't support TLS1.3")
    .with_root_certificates(root_store)
    .with_no_client_auth();

// Finish setup of the rustls connector.
let rustls_connector = hyper_rustls::HttpsConnectorBuilder::new()
    .with_tls_config(config)
    .https_only()
    .enable_http1()
    .enable_http2()
    .build();

// See https://github.com/aws-labs/smithy-rs/discussions/3022 for the
HyperClientBuilder
let http_client = HyperClientBuilder::new().build(rustls_connector);

let shared_conf = aws_config::defaults(BehaviorVersion::latest())
    .http_client(http_client)
    .load()
    .await;

let kms_client = aws_sdk_kms::Client::new(&shared_conf);
let response = kms_client.list_keys().send().await?;

println!("{:?}", response);

Ok(())
}
```

## 부록: SDK에서 사용하는 Rust 상자

이 부록에는에서 사용하는 상자에 대한 고급 정보가 포함되어 있습니다 AWS SDK for Rust. 여기에는 사용하는 Smithy 구성 요소, 특정 빌드 상황에서 사용해야 할 수 있는 상자 및 기타 정보가 포함됩니다.

### Smithy 상자

AWS SDK for Rust 는 대부분의 SDK와 마찬가지로 [Smithy](#)를 기반으로 합니다. AWS SDKs Smithy는 SDK에서 제공하는 데이터 유형 및 함수를 설명하는 데 사용되는 언어입니다. 그런 다음 이러한 모델을 사용하여 SDK 자체를 빌드합니다.

SDK for Rust 상자의 버전과 해당 Smithy 종속성을 살펴볼 때 이러한 상자는 모두 [표준 의미 체계 버전 번호 지정](#)을 사용한다는 점을 알고 있으면 도움이 될 수 있습니다.

Smithy Rust용 상자에 대한 자세한 내용은 [Smithy Rust Design](#)을 참조하세요.

### SDK for Rust와 함께 사용되는 상자

에서 게시한 Smithy 상자는 여러 개 있습니다 AWS. 이 중 일부는 SDK for Rust 사용자와 관련이 있는 반면, 다른 일부는 구현 세부 정보입니다.

#### `aws-smithy-async`

비동기 기능에 Tokio를 사용하지 않는 경우이 크레이트를 포함합니다.

#### `aws-smithy-runtime`

모든 AWS SDKs 포함합니다.

#### `aws-smithy-runtime-api`

SDK에서 사용하는 기본 인터페이스입니다.

#### `aws-smithy-types`

다른 AWS SDKs. 여러 SDKs 사용하는 경우 이를 사용합니다.

#### `aws-smithy-types-convert`

의 내부 및 외부로 이동하기 위한 유틸리티 함수입니다aws-smithy-types.



## 기타 상자

다음 상자는 존재하지만 이에 대해 아무것도 알 필요가 없습니다.

SDK for Rust 사용자에게 필요하지 않은 서버 관련 상자:

- `aws-smithy-http-server`
- `aws-smithy-http-server-python`

SDK 사용자가 사용할 필요가 없는 under-the-hood 코드가 포함된 상자:

- `aws-smithy-checksum-callbacks`
- `aws-smithy-eventstream`
- `aws-smithy-http`
- `aws-smithy-protocol-test`
- `aws-smithy-query`
- `aws-smithy-json`
- `aws-smithy-xml`

지원되지 않고 향후 사라질 운송 상자:

- `aws-smithy-client`
- `aws-smithy-http-auth`
- `aws-smithy-http-tower`

## 문서 기록

이 주제에서는 기록 과정에서 AWS SDK for Rust 개발자 안내서의 중요한 변경 사항에 대해 설명합니다.

변경 사항	설명	날짜
<a href="#">목차 재구성</a>	구성과 사용량을 더 잘 구분하기 위해 목차를 재구성합니다.	2024년 7월 1일
<a href="#">의 일반 가용성 AWS SDK for Rust</a>	SDK의 새로운 일반 가용성 릴리스에 대한 새로운 보안 정보, 새로운 코드 예제 및 업데이트된 코드 예제, 예제를 사용한 단위 테스트에 대한 새로운 세부 정보, 기타 새로운 콘텐츠 및 업데이트된 콘텐츠를 포함하도록 가이드를 업데이트했습니다.	2023년 11월 27일
<a href="#">최소 TLS 버전 적용</a>	SDK에서 TLS 버전을 적용하는 방법에 대한 정보가 추가되었습니다.	2022년 5월 4일
<a href="#">AWS SDK for Rust 개발자 미리 보기 릴리스</a>	<a href="#">개발자 미리 보기 릴리스</a>	2021년 12월 2일

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