

デベロッパーガイド

# AWS SDK for Rust



# AWS SDK for Rust: デベロッパーガイド

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

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスにも関連して、お客様に混乱を招いたり Amazon の信用を傷つけたり失わせたりするいかなる形においても使用することはできません。Amazon が所有していないその他のすべての商標は、Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

# Table of Contents

とは AWS SDK for Rust .....	1
SDK の開始方法 .....	1
SDK メジャーバージョンのメンテナンスとサポート .....	1
追加リソース .....	1
はじめに .....	3
SDK による認証 AWS .....	3
AWS アクセスポータルセッションを開始する .....	4
詳細認証情報 .....	5
Hello チュートリアル .....	6
前提条件 .....	6
最初のSDKアプリを作成する .....	6
基礎 .....	8
前提条件 .....	6
Rust の基礎 .....	8
AWS SDK for Rust 木箱の基礎 .....	10
を操作するためのプロジェクト設定 AWS のサービス .....	10
東京ランタイム .....	11
構成する .....	12
サービスクライアントを作成する .....	12
環境からクライアントを設定する .....	13
サービス固有の設定にビルダーパターンを使用する .....	14
高度な明示的なクライアント設定 .....	15
動作バージョン .....	15
で動作バージョンを設定する Cargo.toml .....	16
コードで動作バージョンを設定する .....	16
認証情報プロバイダ .....	17
認証情報プロバイダーチェーン .....	17
明示的な認証情報プロバイダー .....	20
ID キャッシュ .....	20
AWS リージョン 選択 .....	20
AWS リージョン プロバイダーチェーン .....	20
設定オプション .....	21
クライアントエンドポイント .....	22
カスタム設定 .....	22

例 .....	25
HTTP .....	27
Hyper 1.x .....	27
インターセプター .....	29
インターセプター登録 .....	31
オブザーバビリティ .....	31
ログ記録 .....	32
オペレーション設定を上書きする .....	35
再試行 .....	37
デフォルトの再試行設定 .....	37
最大試行回数 .....	38
遅延とバックオフ .....	38
アダプティブ再試行モード .....	39
タイムアウト .....	39
API タイムアウト .....	40
ストリーム保護の停止 .....	41
SDK を使用する .....	43
リクエストを発行する .....	43
ベストプラクティス .....	44
可能な場合はSDKクライアントを再利用する .....	44
API タイムアウトを設定する .....	45
同時実行 .....	45
用語 .....	45
簡単な例 .....	46
所有権とミュータビリティ .....	47
その他の用語 .....	48
例をより効率的に書き換える (シングルスレッド同時実行) .....	48
例をより効率的に書き換える (マルチスレッド同時実行) .....	51
マルチスレッドアプリケーションのデバッグ .....	52
エラー処理 .....	53
サービスエラー .....	53
エラーメタデータ .....	54
を使用した詳細なエラー印刷 <code>DisplayErrorContext</code> .....	54
Lambda 関数を作成する .....	57
Pagination (ページ分割) .....	57
署名付き を作成する URLs .....	58

署名の基本 .....	59
および POSTPUTリクエストの事前署名 .....	60
スタンドアロン署名者 .....	60
ユニットテスト .....	62
モックオールを使用してモックを自動的に生成する .....	62
静的再生 .....	67
ウェイター .....	71
コードの例 .....	73
API ゲートウェイ .....	74
アクション .....	75
シナリオ .....	76
AWS コミュニティへの貢献 .....	77
API ゲートウェイ管理 API .....	77
アクション .....	75
Application Auto Scaling .....	79
アクション .....	75
Aurora .....	80
基本 .....	82
アクション .....	75
Auto Scaling .....	228
基本 .....	82
アクション .....	75
Amazon Bedrock ランタイム .....	262
Anthropic Claude .....	263
Amazon Cognito Identity Provider .....	278
アクション .....	75
Amazon Cognito Sync .....	279
アクション .....	75
Firehose .....	281
アクション .....	75
Amazon DocumentDB .....	282
サーバーレスサンプル .....	283
DynamoDB .....	285
アクション .....	75
シナリオ .....	76
サーバーレスサンプル .....	283

AWS コミュニティへの貢献 .....	77
Amazon EBS .....	303
アクション .....	75
Amazon EC2 .....	306
基本 .....	82
アクション .....	75
Amazon ECR .....	368
アクション .....	75
Amazon ECS .....	370
アクション .....	75
Amazon EKS .....	373
アクション .....	75
AWS Glue .....	375
基本 .....	82
アクション .....	75
IAM .....	391
基本 .....	82
アクション .....	75
AWS IoT .....	418
アクション .....	75
Kinesis .....	420
アクション .....	75
サーバーレスサンプル .....	283
AWS KMS .....	428
アクション .....	75
Lambda .....	437
基本 .....	82
アクション .....	75
シナリオ .....	76
サーバーレスサンプル .....	283
AWS コミュニティへの貢献 .....	77
MediaLive .....	486
アクション .....	75
MediaPackage .....	487
アクション .....	75
Amazon Polly .....	490

アクション .....	75
シナリオ .....	76
QLDB .....	495
アクション .....	75
Amazon RDS .....	497
サーバーレスサンプル .....	283
Amazon RDS Data Service .....	500
アクション .....	75
Amazon Rekognition .....	502
シナリオ .....	76
Route 53 .....	504
アクション .....	75
Amazon S3 .....	505
基本 .....	82
アクション .....	75
シナリオ .....	76
サーバーレスサンプル .....	283
SageMaker AI .....	556
アクション .....	75
Secrets Manager .....	558
アクション .....	75
Amazon SES API v2 .....	559
アクション .....	75
シナリオ .....	76
Amazon SNS .....	576
アクション .....	75
シナリオ .....	76
サーバーレスサンプル .....	283
Amazon SQS .....	582
アクション .....	75
サーバーレスサンプル .....	283
AWS STS .....	587
アクション .....	75
Systems Manager .....	589
アクション .....	75
Amazon Transcribe .....	591

---

シナリオ .....	76
セキュリティ .....	593
データ保護 .....	593
コンプライアンス検証 .....	594
インフラストラクチャセキュリティ .....	596
最小TLSバージョンを適用する .....	597
で使用する木箱 SDK .....	599
スミティクレート .....	599
で使用する木箱 SDK .....	599
その他のクレート .....	600
ドキュメント履歴 .....	601
.....	dcii



# とは AWS SDK for Rust

Rust は、安全性、速度、同時実行という 3 つの目標に焦点を当てたガベージコレクターのないシステムプログラミング言語です。

AWS SDK for Rust ( SDK) は、Amazon Web Services インフラストラクチャサービスとやり取り APIs するための Rust を提供します。を使用すると SDK、Amazon S3、Amazon 、 DynamoDB EC2 などの上にアプリケーションを構築できます。

トピック

- [SDK の開始方法](#)
- [SDK メジャーバージョンのメンテナンスとサポート](#)
- [追加リソース](#)

## SDK の開始方法

を初めて使用する場合は SDK、「」を読むことから始めることをお勧めします [の使用を開始する AWS SDK for Rust](#)。

へのリクエストを行うためのサービスクライアントを作成および設定する方法など、設定とセットアップについては AWS のサービス、「」を参照してください [を設定する AWS SDK for Rust](#)。

SDK の使用については、『[を使用する AWS SDK for Rust](#)』を参照してください。

Rust コード例の完全なリストについては、「」を参照してください [コードの例](#)。

## SDK メジャーバージョンのメンテナンスとサポート

SDK メジャーバージョンとその基盤となる依存関係のメンテナンスとサポートについては、[AWS SDKs 「およびツールリファレンスガイド」](#) の以下を参照してください。

- [AWS SDKs およびツールメンテナンスポリシー](#)
- [AWS SDKs および ツールバージョンサポートマトリックス](#)

## 追加リソース

このガイドに加えて、以下の SDK 開発者のための貴重なオンラインリソースもあります。

- [AWS SDKs および ツールリファレンスガイド](#): 設定、機能、その他一般的な基本概念が含まれています AWS SDKs。
- [Rust プログラミング言語ウェブサイト](#)
- [AWS SDK for Rust API リファレンス](#)
- [AWS デベロッパーツールブログ](#)
- [のAWS SDK for Rust ソースコード](#) GitHub
- [AWS コードサンプルカタログ](#)

# の使用を開始する AWS SDK for Rust

をインストール、セットアップ、および使用SDKして、AWS リソースにプログラムでアクセスする Rust アプリケーションを作成する方法について説明します。

トピック

- [SDK による認証 AWS](#)
- [の Hello チュートリアル AWS SDK for Rust](#)
- [基礎](#)

## SDK による認証 AWS

で開発 AWS する場合、コードが で認証される方法を確立する必要があります AWS のサービス。AWS リソースへのプログラムによるアクセスは、環境や利用可能な AWS アクセスに応じてさまざまな方法で設定できます。

認証方法を選択し、用に設定するにはSDK、「AWS SDKsおよび ツールリファレンスガイド」の [「認証とアクセス」](#)を参照してください。

ローカルで開発中で、雇用主から認証方法が与えられていない新しいユーザーは、を設定することをお勧めします AWS IAM Identity Center。この方法には、設定を容易に AWS CLI するため、および AWS アクセスポータルに定期的にサインインするための のインストールが含まれます。この方法を選択した場合、「AWS SDKsおよびツールリファレンスガイド」の [IAM Identity Center 認証](#)の手順を完了した後、環境に次の要素が含まれている必要があります。

- アプリケーションを実行する前に AWS アクセスポータルセッションを開始 AWS CLIするために使用する。
- SDK から参照できる設定値のセットを含む [default] プロファイルがある [共有 AWSconfig ファイル](#)。このファイルの場所を確認するには、「AWS SDKsおよびツールリファレンスガイド」の [「共有ファイルの場所」](#)を参照してください。
- 共有 config ファイルは [region](#) 設定を設定します。これにより、AWS リージョンが AWS リクエストSDKに使用するデフォルトが設定されます。このリージョンは、使用するリージョンで指定されていないSDKサービスリクエストに使用されます。
- SDK は、にリクエストを送信する前に、プロファイルの [SSOトークンプロバイダー設定](#)を使用して認証情報を取得します AWS。sso\_role\_name 値は、IAM Identity Center アクセス許可セット

に接続された IAM ロールであり、アプリケーションで AWS のサービス 使用されている へのアクセスを許可します。

次のサンプル config ファイルは、SSO トークンプロバイダー設定で設定されたデフォルトのプロファイルを示しています。プロファイルの `sso_session` 設定は、指定された [sso-session セクション](#) を参照します。sso-session セクションには、AWS アクセスポータルセッションを開始するための設定が含まれています。

```
[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 Rust では、IAM Identity Center 認証を使用するために、アプリケーションに追加のパッケージ (SSO や など SSO0IDC) を追加する必要はありません。

## AWS アクセスポータルセッションを開始する

にアクセスするアプリケーションを実行する前に AWS のサービス、が IAM Identity Center 認証を使用して認証情報を解決 SDK するためのアクティブな AWS アクセスポータルセッションが必要です。設定したセッションの長さに応じて、アクセスは最終的に期限切れになり、SDK で認証エラーが発生します。AWS アクセスポータルにサインインするには、で次のコマンドを実行します AWS CLI。

```
$ aws sso login
```

ガイダンスに従い、デフォルトのプロファイルを設定している場合は、`--profile` オプションを指定してコマンドを呼び出す必要はありません。SSO トークンプロバイダー設定で名前付きプロファイルを使用している場合、コマンドは `aws sso login --profile named-profile` です。

アクティブなセッションがすでにあるかどうかをオプションでテストするには、次の AWS CLI コマンドを実行します。

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

セッションがアクティブな場合、このコマンドへのレスポンスは、共有configファイルで設定された IAM Identity Center アカウントとアクセス許可セットを報告します。

### Note

既にアクティブな AWS アクセスポータルセッションがあり、 を実行している場合はaws sso login、 認証情報を指定する必要はありません。

サインインプロセスにより、データ AWS CLI へのアクセスを許可するように求められる場合があります。AWS CLI は SDK for Python 上に構築されているため、アクセス許可メッセージにはbotocore名前のバリエーションが含まれている可能性があります。

## 詳細認証情報

人間のユーザーとは、別名人間 ID と呼ばれ、人、管理者、デベロッパー、オペレーター、およびアプリケーションのコンシューマーを指します。AWS 環境とアプリケーションにアクセスするには、ID が必要です。組織のメンバーである人間のユーザー、つまり、ユーザーや開発者は、ワークフォースアイデンティティと呼ばれます。

アクセス時に一時的な認証情報を使用します AWS。一時的な認証情報を提供するロールを引き受けることで、人間のユーザーの ID プロバイダーを使用して AWS、アカウントへのフェデレーションアクセスを提供できます。一元的なアクセス管理を行うには、AWS IAM Identity Center (IAM Identity Center) を使用して、アカウントへのアクセスと、それらのアカウント内のアクセス許可を管理することをお勧めします。その他の代替案については、以下を参照してください。

- ベストプラクティスの詳細については、「IAMユーザーガイド」の [「のセキュリティのベストプラクティスIAM」](#) を参照してください。
- 短期 AWS 認証情報を作成するには、「IAMユーザーガイド」の [「一時的なセキュリティ認証情報」](#) を参照してください。
- for Rust でサポートされている他の認証情報プロバイダーについては、「AWS SDKsおよびツールリファレンスガイドSDK」の [「標準化された認証情報プロバイダー」](#) を参照してください。

# の Hello チュートリアル AWS SDK for Rust

## 前提条件

を使用するには AWS SDK for Rust、Rust と Cargo がインストールされている必要があります。

- Rust ツールチェーンのインストール: <https://www.rust-lang.org/tools/install>
- コマンドを実行して cargo-component [ツール](#)をインストールします。 cargo install cargo-component

## 推奨されるツール :

次のオプションツールを にインストールIDEして、コード補完とトラブルシューティングに役立てることができる。

- rust-analyzer 拡張機能については、[Visual Studio Code の「Rust」](#)を参照してください。
- Amazon Q Developer については、[「での Amazon Q Developer 拡張機能またはプラグインIDEのインストール」](#)を参照してください。

## 最初のSDKアプリを作成する

この手順では、DynamoDB テーブルを一覧表示する最初の SDK for Rust アプリケーションを作成します。

1. ターミナルまたはコンソールウィンドウで、アプリケーションを作成するコンピュータ上の場所に移動します。
2. 次のコマンドを実行してhello\_worldディレクトリを作成し、スケルトン Rust プロジェクトを入力します。

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

3. hello\_world ディレクトリに移動し、次のコマンドを使用して必要な依存関係をアプリケーションに追加します。

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

これらの依存関係には、非同期 I/O オペレーションの実装に使用される ク SDKレートなど、DynamoDB の設定機能とサポートを提供するクレートが含まれます。 [tokio](#)

#### Note

Tokio `tokio/full` のような機能を使用しない場合、非同期ランタイムは提供されません。SDK for Rust には非同期ランタイムが必要です。

4. `src` ディレクトリ `main.rs` の を更新して、次のコードを含めます。

```
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**

この例では、結果の最初のページのみを表示します。複数のページの結果を処理する方法については、[the section called “Pagination \(ページ分割\)”](#)「」を参照してください。

5. プログラムを実行します。

```
$ cargo run
```

テーブル名のリストが表示されます。

## 基礎

### 前提条件

を使用するには AWS SDK for Rust、Rust と Cargo がインストールされている必要があります。

- Rust ツールチェーンのインストール: <https://www.rust-lang.org/tools/install>
- コマンドを実行して cargo-component [ツール](#)をインストールします。 `cargo install cargo-component`

### 推奨ツール :

次のオプションツールを にインストールIDEして、コード補完とトラブルシューティングに役立てることができます。

- rust-analyzer 拡張機能については、[Visual Studio Code の「Rust」](#)を参照してください。
- Amazon Q Developer については、[「での Amazon Q Developer 拡張機能またはプラグインIDEのインストール」](#)を参照してください。

## Rust の基礎

以下に、Rust プログラミング言語の基本をいくつか示します。詳細については、[「Rust プログラミング言語」](#)を参照してください。



- Cargo.toml は標準の Rust プロジェクト設定ファイルであり、プロジェクトに関する依存関係とメタデータが含まれています。Rust ソースファイルには .rs ファイル拡張子があります。[「Hello, Cargo!」](#) を参照してください。
- はプロファイルでカスタマイズCargo.tomlできます。[「リリースプロファイルを使用したビルドのカスタマイズ」](#) を参照してください。これらのプロファイルは、共有 AWS config ファイル内のプロファイルの使用とはまったく無関係 AWS であり、独立しています。
- ライブラリの依存関係をプロジェクトとこのファイルに追加する一般的な方法は、を使用することです cargo add。[「cargo-add」](#) を参照してください。
- Rust には次のような基本的な関数構造があります。let キーワードは変数を宣言し、代入 (=) と組み合わせることができます。の後にタイプを指定しない場合let、コンパイラはタイプを推測します。[「変数とミュータビリティ」](#) を参照してください。

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

- 明示的な型 x を持つ変数を宣言するためにT、Rust は構文 を使用しますx: T。[「データ型」](#) を参照してください。
- struct X {} は新しいタイプ を定義しますX。メソッドは、カスタム構造体タイプ に実装されますX。タイプのメソッドXは、キーワード のプレフィックスが付いた実装ブロックで宣言されますimpl。実装ブロック内で、 はメソッドが呼び出された構造体のインスタンスselfを参照します。[キーワードimplとメソッドの構文](#) を参照してください。
- 感嘆符 (「!」) が関数定義または関数呼び出しのように見えるものに従う場合、コードはマクロを定義または呼び出しています。[マクロ](#) を参照してください。
- Rust では、回復不可能なエラーはpanic!マクロによって表されます。プログラムがにpanic!遭遇すると、実行が停止し、失敗メッセージを印刷し、解凍し、スタックをクリーンアップして、終了します。[「による回復不可能なエラーpanic!」](#) を参照してください。
- Rust は、他のプログラミング言語のように基本クラスからの機能の継承をサポートしていません。Rust traits がメソッドのオーバーロードを提供する方法です。特性は、概念的にインターフェイスに似ていると考えることができます。ただし、特性と真のインターフェイスには違いがあり、多くの場合、設計プロセスでは異なる方法で使用されます。[「特性: 共有動作の定義」](#) を参照してください。
- 多型とは、それぞれを個別に記述することなく、複数のデータ型の機能をサポートするコードです。Rust は、列挙型、特性、ジェネリックを通じて多型をサポートしています。[「タイプシステムとしての継承」](#) と [「コード共有としての継承」](#) を参照してください。

- Rust はメモリについて非常に明示的です。スマートポインタ「はポインタのように動作するデータ構造ですが、追加のメタデータと機能もあります」。 [「スマートポインタ」](#)を参照してください。
- タイプCowは、clone-on-write必要に応じて発信者にメモリ所有権を転送できるスマートポインタです。「[Enum std::borrow::Cow](#)」を参照してください。
- タイプArcは、割り当てられたインスタンスをカウントするアトミックリファレンスカウントスマートポインタです。「[Struct std::sync::Arc](#)」を参照してください。
- SDK for Rust は、複雑な型を構築するためにビルダーパターンを頻繁に使用します。

## AWS SDK for Rust 木箱の基礎

- SDK for Rust ライブラリは、それぞれ異なるライブラリクレートに分割されます AWS のサービス。これらのクレートは <https://docs.rs/> で入手できます。
- AWS のサービス クレートはaws-sdk-*[servicename]*、aws-sdk-s3や などの の命名規則に従いますaws-sdk-dynamodb。
- SDK for Rust 機能のプライマリコアクレートは aws-configです。これは、環境から設定を読み取る機能を提供するため、ほとんどのプロジェクトに含まれています。
  - これを AWS のサービス と呼ばれる と混同しないでください AWS Config。これはサービスであるため、標準の規則に従い、 と呼ばれますaws-sdk-config。

## を操作するためのプロジェクト設定 AWS のサービス

- AWS のサービス アプリケーションで使用する各 のプロジェクトに木箱を追加する必要があります。
- クレートを追加する推奨方法はcargo add *[crateName]*、 などの を実行して、プロジェクトのディレクトリのコマンドラインを使用することでcargo add aws-sdk-s3。
  - これにより、 のCargo.toml下のプロジェクトの に行が追加されます[dependencies]。
  - デフォルトでは、これにより最新バージョンの木箱がプロジェクトに追加されます。
- ソースファイルで、 useステートメントを使用して、クレートから項目をスコープ内に取り込みます。Rust プログラミング言語ウェブサイトの [「外部パッケージの使用」](#)を参照してください。
  - 木箱名はハイフンで囲まれることがよくありますが、実際に木箱を使用すると、ハイフンはアンダースコアに変換されます。例えば、aws-configクレートはコードuseステートメントとして使用されますuse aws\_config。

- 設定は複雑なトピックです。設定はコードで直接行うか、環境変数または設定ファイルで外部で指定できます。詳細については、「[設定オプション](#)」を参照してください。
- が設定をSDKロードすると、ほとんどの設定に合理的なデフォルトがあるため、実行を停止する代わりに無効な値がログに記録されます。ログ記録を有効にする方法については、「」を参照してください[AWS SDK for Rust コードのログ記録を有効にする](#)。
- ほとんどの環境変数と設定ファイル設定は、プログラムの開始時に 1 回ロードされます。値の更新は、プログラムを再起動するまで表示されません。

## 東京ランタイム

- Tokio は、SDK for Rust プログラミング言語の非同期ランタイムであり、asyncタスクを実行します。[tokio.rs](https://tokio.rs) および [docs.rs/tokio](https://docs.rs/tokio) を参照してください。
- SDK for Rust には非同期ランタイムが必要です。プロジェクトに次の木箱を追加することをお勧めします。

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

- `tokio::main` 属性マクロは、プログラムへの非同期メインエントリーポイントを作成します。このマクロを使用するには、次に示すように、`main`メソッドの前の行に追加します。

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

# を設定する AWS SDK for Rust

AWS SDK for Rustを設定する方法について説明します。で開発 AWS するときに、コードが で認証される方法を確認する必要があります AWS のサービス。AWS リージョン 使用する も設定する必要があります。

[AWS SDKs および ツールリファレンスガイド](#)には、の多くに共通する設定、機能、その他の基本的な概念も含まれています AWS SDKs。

## トピック

- [サービスクライアントを作成する](#)
- [動作バージョン](#)
- [認証情報プロバイダー](#)
- [AWS リージョン 選択](#)
- [設定オプション](#)
- [クライアントエンドポイント](#)
- [HTTP](#)
- [インターセプター](#)
- [オブザーバビリティ機能を使用する](#)
- [クライアントの1つのオペレーション設定を上書きする](#)
- [再試行](#)
- [タイムアウト](#)

## サービスクライアントを作成する

にリクエストを行うには AWS のサービス、まずそのサービスのクライアントをインスタンス化します。タイムアウト、クライアント、再試行設定など、サービスHTTPクライアントの共通設定を構成できます。

各サービスクライアントには、AWS リージョン と認証情報プロバイダーが必要です。SDK は、これらの値を使用して、リソースの正しいリージョンにリクエストを送信し、正しい認証情報でリクエストに署名します。これらの値は、プログラムでコードで指定することも、環境から自動的にロードすることもできます。

**Note**

サービスクライアントは構築にコストがかかる場合があり、一般的に共有することを意図しています。これを容易にするために、すべてのClient構造体は `Clone` を実装します。

SDK には、設定の値を見つけるためにチェックする一連の場所 (またはソース) があります。

1. コードまたはサービスクライアント自体に設定されている明示的な設定は、他の設定よりも優先されます。
2. 環境変数
  - 環境変数の設定の詳細については、「AWS SDKsおよびツールリファレンスガイド」の「[環境変数](#)」を参照してください。
3. 共有 config および credentials ファイル
  - これらのファイルの設定の詳細については、「[configおよびツールリファレンスガイド](#)」の「[共有 credentials ファイルと ファイル](#)」を参照してください。AWS SDKs
4. SDK ソースコード自体によって提供されるデフォルト値が最後に使用されます。
  - リージョンなどの一部のプロパティにはデフォルトがありません。コード、環境設定、または共有configファイルで明示的に指定する必要があります。が必要な設定を解決SDKできない場合、実行時にAPIリクエストが失敗する可能性があります。

ほとんどの環境変数設定と config および credentials ファイル設定は、一貫した動作のために複数の AWS SDKs および ツールによって共有されます。が環境変数または設定ファイルから解決 SDK できるすべての設定を確認するには、「[およびツールリファレンスガイド](#)」の「[設定リファレンス](#)」を参照してください。AWS SDKs

## 環境からクライアントを設定する

環境ソース設定でクライアントを作成するには、aws-configクレートの静的メソッドを使用します。

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

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

この方法でクライアントを作成すると、Amazon Elastic Compute Cloud で実行する場合や AWS Lambda、サービスクライアントの設定を環境から直接利用できるその他のコンテキストで実行する場合に便利です。これにより、コードが実行中の環境から切り離され、コードを変更 AWS リージョン せずにアプリケーションを複数の にデプロイすることが容易になります。

特定のプロパティを明示的に上書きできます。明示的な設定は、実行環境から解決された設定よりも優先されます。次の例では、環境から設定をロードしますが、明示的に を上書きします AWS リージョン。

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

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

#### Note

すべての設定値が作成時にクライアントによって取得されるわけではありません。一時的なアクセスキーや IAM Identity Center 設定などの認証情報関連の設定は、クライアントを使用してリクエストを行うときに、認証情報プロバイダーレイヤーによってアクセスされます。

前の例BehaviorVersion::latest()に示すコードは、デフォルトSDKに使用する のバージョンを示しています。BehaviorVersion::latest()は、ほとんどの場合に適しています。詳細については、「[動作バージョン](#)」を参照してください。

## サービス固有の設定にビルダーパターンを使用する

特定のサービスクライアントタイプでのみ設定できるオプションがいくつかあります。ただし、ほとんどの場合、環境から設定の大部分をロードし、特に追加オプションを追加する必要があります。ビルダーパターンは、AWS SDK for Rust クレート内の一般的なパターンです。を使用して一般設定をロードしaws\_config::defaults、次に fromメソッドを使用して、作業しているサービスのビルダーにその設定をロードします。その後、そのサービスの一意の設定値を設定し、 を呼び出すことができますbuild。最後に、クライアントはこの変更後の設定から作成されます。

```
// 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);
```

特定のタイプのサービスクライアントで使用できる追加の方法を検出する 1 つの方法は、 の など、APIドキュメントを使用することで [aws\\_sdk\\_s3::config::Builder](#)。

## 高度な明示的なクライアント設定

環境から設定をロードする代わりに特定の値を使用してサービスクライアントを設定するには、次に示すように、クライアントConfigビルダーで指定できます。

```
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);
```

でサービス設定を作成すると `aws_sdk_s3::Config::builder()`、デフォルト設定はロードされません。デフォルトは、 に基づいて設定を作成する場合にのみロードされま  
ず `aws_config::defaults`。

特定のサービスクライアントタイプでのみ設定できるオプションがいくつかあります。前の例は、Amazon S3 クライアントで `endpoint_resolver` 関数を使用して、この例を示しています。

## 動作バージョン

AWS SDK for Rust デベロッパーは、言語とその主要なライブラリが提供する堅牢で予測可能な動作を期待し、それに依存しています。SDK for Rust を使用するデベロッパーが期待される動作を取得できるようにするには、クライアント設定に を含める必要があります `BehaviorVersion`。は、デフォルトSDKが予期される のバージョン `BehaviorVersion` を指定します。これにより、 は時間の経過とともにSDK進化し、新しい標準に合わせてベストプラクティスを変更し、アプリケーションの動作に予期しない悪影響を与えることなく新機能をサポートします。



**⚠ Warning**

を明示的に指定せずに を設定したり、クライアントSDKを作成しようとする  
とBehaviorVersion、コンストラクタは panic.

例えば、新しいバージョンの SDKが新しいデフォルトの再試行ポリシーでリリースされたとします。アプリケーションが の以前のバージョンBehaviorVersionと一致する を使用している場合 SDK、その以前の設定が新しいデフォルト設定の代わりに使用されます。

SDK for Rust の新しい動作バージョンがリリースされるたびに、前の BehaviorVersion は SDK for Rust deprecated 属性でマークされ、新しいバージョンが追加されます。これにより、コンパイル時に警告が発生しますが、それ以外の場合はビルドを通常どおりに続行できます。BehaviorVersion::latest()も更新され、新しいバージョンのデフォルト動作が示されます。

**i Note**

ほとんどの場合、コードBehaviorVersion::latest()で を使用するか、Cargo.toml  
ファイルの機能フラグbehavior-version-latestを使用する必要があります。必要な場  
合にのみ、特定のバージョンに固定することをお勧めします。

## で動作バージョンを設定する Cargo.toml

ファイルで適切な機能フラグを含めるaws-sdk-iamことで、SDKおよび aws-sdk-s3や などの  
個々のモジュールの動作バージョンを指定できますCargo.toml。現時点では、latestのバージョ  
ンのみ がサポートSDKされていますCargo.toml。

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

## コードで動作バージョンを設定する

コードは、SDKまたはクライアントを設定するときに指定することで、必要に応じて動作バージョ  
ンを変更できます。

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



この例では、環境を使用してを設定する設定を作成しますSDKが、BehaviorVersionを に設定しますv2023\_11\_09()。

## 認証情報プロバイダー

AWS を使用してにリクエストを実行するために AWS SDK for Rust、は によって発行された暗号署名付き認証情報SDKを使用します AWS。実行時に、は複数の場所をチェックして認証情報の設定値SDKを取得します。

取得した設定に[AWS IAM Identity Center シングルサインオンアクセス設定](#)が含まれている場合、SDKは IAM Identity Center と連携して、リクエストに使用する一時的な認証情報を取得します AWS のサービス。

取得した設定に[一時的な認証情報](#)が含まれている場合、SDKはそれらを使用して AWS のサービス呼び出しを行います。一時的な認証情報は、アクセスキーとセッショントークンで構成されます。

による認証 AWS は、コードベース外で処理できます。認証情報プロバイダーチェーンSDKを使用して、によって多くの認証方法を自動的に検出、使用、更新できます。

プロジェクトの AWS 認証を開始するためのガイド付きオプションについては、「AWS SDKsおよびツールリファレンスガイド」の[「認証とアクセス」](#)を参照してください。

## 認証情報プロバイダーチェーン

クライアントの構築時に認証情報プロバイダーを明示的に指定しない場合、SDK for Rust は認証情報プロバイダーチェーンを使用して、認証情報を提供できる一連の場所を確認します。がこれらの場所のいずれかで認証情報SDKを検出すると、検索は停止します。クライアントの構築の詳細については、「」を参照してください[サービスクライアントを作成する](#)。

次の例では、コードに認証情報プロバイダーを指定していません。SDK は、認証情報プロバイダーチェーンを使用して、ホスティング環境で設定された認証を検出し、その認証を呼び出しに使用します AWS のサービス。

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

## 認証情報の取得順序

認証情報プロバイダーチェーンは、次の事前定義されたシーケンスを使用して認証情報を検索します。

## 1. キー環境変数にアクセスする

は、`AWS_ACCESS_KEY_ID`と`AWS_SECRET_ACCESS_KEY`、および`AWS_SESSION_TOKEN`環境変数から認証情報をロードSDKしようとしています。

## 2. 共有 AWS `config` ファイルと `credentials` ファイル

は、共有 AWS `config` ファイルと `credentials` ファイル内の`[default]`プロファイルから認証情報をロードSDKしようとしています。`AWS_PROFILE` 環境変数を使用して、を使用する代わりにでロードSDKする名前付きプロファイルを選択できます`[default]`。`config` および `credentials` ファイルは、さまざまな AWS SDKs およびツールで共有されます。これらのファイルの詳細については、「[configおよびツールリファレンスガイド](#)」の「[共有credentialsファイル](#)」と「」を参照してください。AWS SDKs

IAM Identity Center を使用して認証する場合、SDK for Rust が `aws sso login` コマンドを実行して AWS CLI セットアップされたシングルサインオントークンを使用する場合があります。は、IAM アイデンティティセンターが有効なトークンと交換した一時的な認証情報SDKを使用します。SDK 次に、は呼び出し時に一時的な認証情報を使用します AWS のサービス。このプロセスの詳細については、「[AWS SDKsおよびツールリファレンスガイド](#)」の「[のSDK認証情報解決 AWS のサービス](#)を理解する」を参照してください。

- このプロバイダーの設定に関するガイダンスについては、「[AWS SDKsおよびツールリファレンスガイド](#)」のIAM「[アイデンティティセンター認証](#)」を参照してください。
- このプロバイダーSDKの設定プロパティの詳細については、「[AWS SDKsおよびツールリファレンスガイド](#)」のIAM「[Identity Center 認証情報プロバイダー](#)」を参照してください。

## 3. AWS STS ウェブ ID

アクセスを必要とするモバイルアプリケーションまたはクライアントベースのウェブアプリケーションを作成する場合 AWS、AWS Security Token Service (AWS STS) は、パブリック ID プロバイダー (IdP) を介して認証されたフェデレーティッドユーザーの一時的なセキュリティ認証情報のセットを返します。

- プロファイルでこれを指定すると、SDKまたはツールは `AssumeRoleWithWebIdentity` API メソッドを使用して AWS STS 一時的な認証情報の取得を試みます。この方法の詳細については、「[AWS Security Token Service APIリファレンス](#)」の[AssumeRoleWithWebIdentity](#)「」を参照してください。
- このプロバイダーの設定に関するガイダンスについては、「[AWS SDKsおよびツールリファレンスガイド](#)」の「[ウェブ ID または OpenID Connect とのフェデレーション](#)」を参照してください。

- このプロバイダーSDKの設定プロパティの詳細については、「AWS SDKsおよびツールリファレンスガイド」の[「ロール認証情報プロバイダーを引き受ける」](#)を参照してください。

#### 4. Amazon ECSおよび Amazon EKSコンテナの認証情報

Amazon Elastic Container Service タスクと Kubernetes サービスアカウントには、IAMロールを関連付けることができます。IAM ロールで付与されるアクセス許可は、ポッドのタスクまたはコンテナで実行されているコンテナによって引き受けられます。このロールにより、SDK for Rust アプリケーションコード (コンテナ上) が他のを使用できるようになります AWS のサービス。

は、AWS\_CONTAINER\_CREDENTIALS\_RELATIVE\_URIまたはAWS\_CONTAINER\_CREDENTIALS\_FULL\_URI環境変数から認証情報を取得SDKしようとしています。これは、Amazon ECSおよび Amazon によって自動的に設定できますEKS。

- Amazon のこのロールの設定の詳細についてはECS、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon [ECSタスクIAMロール](#)」を参照してください。
- Amazon EKSのセットアップ情報については、「[Amazon ユーザーガイド](#)」の「[Amazon EKS Pod Identity Agent のセットアップ](#)」を参照してください。 EKS
- このプロバイダーSDKの設定プロパティの詳細については、「AWS SDKsおよびツールリファレンスガイド」の[「コンテナ認証情報プロバイダー」](#)を参照してください。

#### 5. Amazon EC2 インスタンスメタデータサービス

IAM ロールを作成し、インスタンスにアタッチします。インスタンス上の SDK for Rust アプリケーションは、ロールによって提供された認証情報をインスタンスメタデータから取得しようとしています。

- SDK for Rust は のみをサポートします[IMDSv2](#)。
- このロールの設定とメタデータの使用の詳細については、[IAM 「Amazon ユーザーガイド」の「Amazon のロールEC2」](#)および[「インスタンスメタデータの操作」](#)を参照してください。EC2
- このプロバイダーSDKの設定プロパティの詳細については、「AWS SDKsおよびツールリファレンスガイド」の[IMDS 「 認証情報プロバイダー」](#)を参照してください。

6. この時点で認証情報がまだ解決されない場合は、オペレーション panics でエラーが発生します。

AWS 認証情報プロバイダーの設定の詳細については、「AWS SDKsおよびツールリファレンスガイド」の[「標準化された認証情報プロバイダー」](#)を参照してください。

## 明示的な認証情報プロバイダー

認証情報プロバイダーチェーンを使用して認証方法を検出する代わりに、がSDK使用する特定の認証情報プロバイダーを指定できます。を使用して一般的な設定をロードする場合aws\_config::defaults、次に示すように、カスタム認証情報プロバイダーを指定できます。

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

[ProvideCredentials](#) 特性を実装することで、独自の認証情報プロバイダーを実装できます。

## ID キャッシュ

SDK は、認証情報とSSOトークンなどの他の ID タイプをキャッシュします。デフォルトでは、最初のリクエスト時に認証情報をロードし、キャッシュしてから、有効期限が近い別のリクエスト中に認証情報の更新を試みる遅延キャッシュ実装SDKを使用します。同じ から作成されたクライアントSdkConfigは、 を共有します[IdentityCache](#)。

## AWS リージョン 選択

を使用して、特定の地域 AWS のサービス で動作する にアクセスできます AWS リージョン。これは、冗長性と、データやアプリケーションがユーザーとユーザーがアクセスする場所に近接して実行されるように維持するためにも役立ちます。リージョンの使用の詳細については、「[AWS リージョン](#) AWS SDKsおよびツールリファレンスガイド」の「」を参照してください。

### Important

ほとんどのリソースは特定の に存在し AWS リージョン、 を使用するときにはリソースに正しいリージョンを指定する必要がありますSDK。

## AWS リージョン プロバイダーチェーン

次のルックアッププロセスは、実行環境からサービスクライアントの設定をロードするとき 사용됩니다。SDK 検出結果セットの最初の値は、クライアントの設定で使用されます。サービスクライアントの作成の詳細については、「」を参照してください[環境からクライアントを設定する](#)。

1. プログラムで設定された明示的なリージョン。
2. `AWS_REGION` 環境変数が確認されます。
  - AWS Lambda サービスを使用している場合、この環境変数は AWS Lambda コンテナによって自動的に設定されます。
3. 共有 AWS configファイルの `region` プロパティがチェックされます。
  - `AWS_CONFIG_FILE` 環境変数を使用して、共有configファイルの場所を変更できます。このファイルの保存場所の詳細については、「[configおよびツールリファレンスガイド](#)」の「[共有ファイルと credentials ファイルの場所](#)」を参照してください。AWS SDKs
  - `AWS_PROFILE` 環境変数を使用して、デフォルトの代わりに名前付きプロファイルを選択できます。さまざまなプロファイルの設定の詳細については、「[configおよびツールリファレンスガイド](#)」の「[共有ファイルと credentials ファイル](#)」を参照してください。AWS SDKs
4. は、Amazon EC2 Instance Metadata Service を使用して、現在実行中の Amazon EC2インスタンスのリージョンを決定SDKしようとしています。
  - は AWS SDK for Rust のみをサポートしますIMDSv2。

## 設定オプション

多くの設定はコードの外部で処理できます。ほとんどの設定は、環境変数として設定することも、別の共有 AWS configファイルで設定することもできます。共有configファイルは、プロファイルと呼ばれる個別の設定セットを維持し、異なる環境やテストに異なる設定を提供できます。

環境変数と共有configファイル設定は標準化され、おおよび ツール間で AWS SDKs共有され、さまざまな言語で一貫した機能をサポートします。

これらのメソッドを使用したアプリケーションの設定と、各クロス SDK 設定の詳細については、「」およびAWS SDKs「[ツールリファレンスガイド](#)」を参照してください。対象ページには次のようなものがあります。

- [共有 config ファイルと credentials ファイル](#) - プロファイルとこれらの設定ファイルの形式について説明します。
- [共有 ファイルconfigと credentials ファイルの場所](#) - これらのファイルのデフォルトの場所と変更方法について説明します。
- [環境変数のサポート](#) - 環境変数の設定について説明します。
- [リファレンスの設定](#) - すべてのクロスSDK設定のリファレンス情報。

# クライアントエンドポイント

が AWS SDK for Rust を呼び出すとき AWS のサービス、最初のステップの 1 つは、リクエストをルーティングする場所を決定することです。このプロセスはエンドポイント解決と呼ばれます。

サービスクライアントを作成する SDK ときに、 のエンドポイント解決を設定できます。エンドポイント解決のデフォルト設定では通常問題ありませんが、デフォルト設定を変更する理由はいくつかあります。2 つの理由の例を以下に示します。

- サービスのプレリリースバージョンまたはサービスのローカルデプロイにリクエストを行うには。
- でまだモデル化されていない特定のサービス機能にアクセスするには SDK。

## ⚠ Warning

エンドポイント解決は高度な SDK トピックです。デフォルト設定を変更すると、コードが破損するリスクがあります。デフォルト設定は、本番環境のほとんどのユーザーに適用されます。

カスタムエンドポイントは、すべてのサービスリクエストに使用されるようにグローバルに設定することも、特定の にカスタムエンドポイントを設定することもできます AWS のサービス。

カスタムエンドポイントは、環境変数または共有 AWS config ファイルの設定を使用して設定できます。このアプローチの詳細については、「AWS SDKs およびツールリファレンスガイド」の「[サービス固有のエンドポイント](#)」を参照してください。すべての の共有 config ファイル設定と環境変数の完全なリストについては AWS のサービス、[「サービス固有のエンドポイントの識別子」](#)を参照してください。

または、次のセクションに示すように、このカスタマイズをコードで設定することもできます。

## カスタム設定

サービスクライアントのエンドポイント解決は、クライアントの構築時に使用できる 2 つの方法でカスタマイズできます。

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



両方のプロパティを設定できます。ただし、ほとんどの場合、1つだけ指定します。一般的な使用のために、`endpoint_url`は最も頻繁にカスタマイズされます。

## エンドポイントの設定 URL

の値を設定`endpoint_url`して、サービスの「ベース」ホスト名を指定できます。ただし、この値はクライアントの`ResolveEndpoint`インスタンスにパラメータとして渡されるため、最終値ではありません。その後、`ResolveEndpoint`実装はその値を検査して変更し、最終的なエンドポイントを決定できます。

## エンドポイントリゾルバーの設定

サービスクライアントの`ResolveEndpoint`実装によって、特定のリクエストSDKに使用する最終的な解決済みエンドポイントが決まります。サービスクライアントは、すべてのリクエストに対して`resolve_endpoint`メソッドを呼び出し、リゾルバーから返された[EndpointFuture](#)値を使用します。それ以上の変更はありません。

次の例は、ステージングや本番稼働など、ステージごとに異なるエンドポイントを解決する Amazon S3 クライアントにカスタムエンドポイントリゾルバー実装を提供する方法を示しています。

```
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

エンドポイントリゾルバー、さらにはResolveEndpoint特性は各サービスに固有であるため、サービスクライアント設定でのみ設定できます。一方URL、エンドポイントは、共有設定(そこから派生したすべてのサービスに適用)を使用するか、特定のサービスに対して設定できます。

## ResolveEndpoint パラメータ

resolve\_endpoint メソッドは、エンドポイント解決で使用されるプロパティを含むサービス固有のパラメータを受け入れます。

すべてのサービスには、次の基本プロパティが含まれています。

名前	型	説明
region	文字列	クライアントの AWS リージョン
endpoint	文字列	の値セットの文字列表現 endpointUrl
use_fips	ブール値	クライアントの設定でFIPSエンドポイントが有効になっているかどうか
use_dual_stack	ブール値	クライアントの設定でデュアルスタックエンドポイントが有効になっているかどうか

AWS のサービスでは、解決に必要な追加のプロパティを指定できます。例えば、Amazon S3 [エンドポイントパラメータ](#)には、バケット名と Amazon S3-specific機能設定がいくつか含まれています。例えば、force\_path\_styleプロパティは、仮想ホストアドレス指定を使用できるかどうかを決定します。

独自のプロバイダーを実装する場合、エンドポイントパラメータの独自のインスタンスを作成する必要はありません。SDK は、各リクエストのプロパティを提供し、の実装に渡します resolve\_endpoint。



## の使用endpoint\_urlと の使用の比較 endpoint\_resolver

を使用する設定endpoint\_urlと を使用する設定の 2 つが、同等のエンドポイント解決動作でクライアントendpoint\_resolverNOTを生成することを理解することが重要です。

```
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();
```

を設定するクライアントURLは、エンドポイント解決の一部として変更できる (デフォルト) プロバイダーに渡されるベースendpoint\_urlを指定します。

を設定するクライアントは、Amazon S3 クライアントURLが使用する最終的なendpoint\_resolverを指定します。

## 例

カスタムエンドポイントは、多くの場合テストに使用されます。クラウドベースのサービスを呼び出す代わりに、呼び出しはローカルにホストされ、シミュレートされたサービスにルーティングされます。このようなオプションは 2 つあります。

- [DynamoDB local](#) – Amazon DynamoDB サービスのローカルバージョン。
- [LocalStack](#) – ローカルマシンのコンテナで実行されるクラウドサービスエミュレータ。

次の例は、これら 2 つのテストオプションを使用するカスタムエンドポイントを指定する 2 つの異なる方法を示しています。

## DynamoDB local をコード内で直接使用する

前のセクションで説明したように、をコードで `endpoint_url` 直接設定して、ローカル DynamoDB サーバーを指すように基本エンドポイントを上書きできます。コードでは、次の操作を行います。

```
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);
```

[完全な例](#)については、「」を参照してください GitHub。

## config ファイル LocalStack を使用して を使用する

共有 AWS config ファイルで [サービス固有のエンドポイント](#) を設定できます。次の設定プロファイルは `endpoint_url`、ポート `localhost` でに接続するように を設定します 4566。LocalStack 設定の詳細については、ドキュメントウェブサイトの「[エンドポイント LocalStack を介したアクセス URL](#)」を参照してください。LocalStack

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

SDK は、共有 config ファイルの変更を取得し、`localstack` プロファイルを使用するときに SDK クライアントに適用します。このアプローチを使用すると、コードにエンドポイントへの参照を含める必要がなく、次のようになります。

```
// 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);
```

[完全な例](#)については、「」を参照してください GitHub。

## HTTP

for Rust AWS SDKには、以下の HTTP関連の設定があります。

トピック

- [Hyper 1.x](#)

## Hyper 1.x

### Note

AWS SDK for Rust は、安定した 1.0 バージョンをhyperリリースする前に開発されました。SDK 現在、はデフォルトで v0.14 バージョンの を使用していますhyper。

hyper は Rust 用の下位レベルのHTTPライブラリで、 でサービスAPI呼び出し AWS SDK for Rust を行うことができます。このクレートのリファレンスドキュメントについては、「[hyper - Rust](#)」を参照してください。デフォルトでは、 SDK for Rust は v0.14 バージョンの hyperとCryptoMode:Ringプロバイダーを使用します。

hyper 1.0 をサポートするクライアントは[aws-smithy-experimental](#)、クレートにあります。

## 暗号プロバイダーを選択する

を選択しCryptoProvider、aws-smithy-experimentalクレートで必要な機能を有効にする必要があります。

次のCryptoProviderモードを使用できます。

- `CryptoMode:AwsLc` - に基づく暗号化プロバイダー [aws-lc-rs](#)。これは、Google Boring AWS SSL プロジェクト AWS と OpenSSL プロジェクトのコードに基づいて、Cryptography チームがとその顧客のために管理する汎用暗号化ライブラリです。
  - 機能が必要です `crypto-aws-lc`。
- `CryptoMode:AwsLcFips` - に基づく FIPS 準拠の暗号化プロバイダー [aws-lc-rs](#)。
  - 機能が必要です `crypto-aws-lc-fips`。

**Note**

これには、コンパイルできる追加のビルドツールが必要になる場合があります。詳細については、[aws-lc-rs](#) リポジトリとビルド手順を参照してください。

- `CryptoMode:Ring` - に基づく暗号化プロバイダー [ring](#)。これは「Rust を使用した安全で高速、小さな暗号」です。
  - 機能が必要です `crypto-ring`。

**Note**

これは、SDK for Rust がデフォルトの v0.14 hyper クライアントで使用する暗号化プロバイダーです。

木箱の機能を有効にする方法

次の例は、プロジェクトの `aws-smithy-experimental` の木箱に `crypto-aws-lc` 機能を追加する方法を示しています `Cargo.toml`。

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

または、プロジェクトのフォルダのコマンドプロンプトで次のコマンドを実行できます。

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

詳細については、「The Cargo Book」の「[Features](#)」を参照してください。

## 1.0 hyper クライアントを作成し、で使用する SDK

hyper 1.0 クライアントを作成するには、プロジェクトの `Cargo.toml` に `aws-smithy-experimental` 木箱 `Cargo.toml` を追加します。HyperClientBuilder では、を選択する必要があります。CryptoProvider。前のセクションでは、プロバイダーの選択と、クレートの暗号関連の機能の `aws-smithy-experimental` を有効にする方法について説明します。

次のコードでは、Amazon Simple Storage Service (Amazon S3) サービス HTTP クライアントで使用する hyper 1.0 クライアントの暗号化プロバイダー `CryptoMode::AwsLc` としてを使用します。Amazon S3 この例では、プロジェクトの `Cargo.toml` に AWS のサービスの木箱 `aws-sdk-s3` も含める必要があります。

```
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 クライアントは、共有 config 構造体または特定のサービスの設定構造体で設定できます。

## インターセプター

インターセプターを使用して、API リクエストとレスポンスの実行にフックできます。インターセプターは、リクエスト/レスポンスのライフサイクルに動作を挿入するために記述したコードを SDK 呼び出すオープンエンドメカニズムです。これにより、処理中のリクエストの変更、リクエスト処理のデバッグ、エラーの表示などを行うことができます。

次の例は、再試行ループに入る前にすべての送信リクエストにヘッダーを追加するシンプルなインターセプターを示しています。

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

詳細と使用可能なインターセプターフックについては、[インターセプト](#)特性を参照してください。

## インターセプター登録

インターセプターは、サービスクライアントを構築するとき、または特定のオペレーションの設定を上書きするときに登録します。での登録は、インターセプターをクライアントのすべてのオペレーションに適用するか、特定のオペレーションのみに適用するかによって異なります。

### サービスクライアント上のすべてのオペレーションのインターセプター

クライアント全体のインターセプターを登録するには、Builderパターンを使用してインターセプターを追加します。

```
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);
```

### 特定のオペレーションのみのインターセプター

1つのオペレーションのみのインターセプターを登録するには、customize拡張機能を使用します。この方法を使用して、オペレーションごとのレベルでサービスクライアント設定を上書きできます。カスタマイズ可能なオペレーションの詳細については、「」を参照してください[クライアントの1つのオペレーション設定を上書きする](#)。

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

## オブザーバビリティ機能を使用する

オブザーバビリティとは、システムの現在の状態を、それが出力するデータから推測できる範囲です。出力されるデータは、一般的にテレメトリと呼ばれます。

## トピック

- [AWS SDK for Rust コードのログ記録を有効にする](#)

## AWS SDK for Rust コードのログ記録を有効にする

は、ログ記録に [トレース](#) フレームワーク AWS SDK for Rust を使用します。

1. プロジェクトディレクトリのコマンドプロンプトで、[トレースサブスクリプト](#) を依存関係として追加します。

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

これにより、Cargo.toml ファイルの [dependencies] セクションに本箱が追加されます。

2. サブスクリバラーを初期化します。通常、これは for SDK Rust オペレーションを呼び出す前に main、関数の早い段階で行われます。

```
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 環境変数を使用してログ記録を有効にします。ログ情報の表示を有効にするには、コマンドプロンプトで、RUST\_LOG 環境変数をログインするレベルに設定します。次の例では、ログ記録を debug レベルに設定します。



## Linux/macOS

```
$ RUST_LOG=debug
```

## Windows

を使用している場合VSCode、ターミナルウィンドウのデフォルトは `PowerShell`。使用しているプロンプトのタイプを確認します。

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

## PowerShell

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

### 4. プログラムを実行します。

```
$ cargo run
```

コンソールまたはターミナルウィンドウに追加の出力が表示されます。

詳細については、`tracing-subscriber`ドキュメントの [「環境変数を使用したイベントのフィルタリング」](#) を参照してください。

## ログ出力の解釈

前のセクションの手順に従ってログ記録を有効にすると、デフォルトで追加のログ情報が標準出力に出力されます。

デフォルトのログ出力形式 (トレースモジュールでは「full」と呼ばれます) を使用している場合、ログ出力に表示される情報は次のようになります。

```
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
```

各エントリには以下が含まれます。

- ログエントリのタイムスタンプ。
- エントリのログレベル。これは、INFO、DEBUGなどの単語ですTRACE。
- ログエントリが生成されたネストされたスパンのセット。コロン(=)で区切られます。これにより、ログエントリのソースを識別できます。
- ログエントリを生成したコードを含む Rust モジュールパス。
- ログメッセージテキスト。

トレースモジュールの標準出力形式は、ANSIエスケープコードを使用して出力を色分けします。出力をフィルタリングまたは検索するときは、これらのエスケープシーケンスに注意してください。

#### Note

ネストされた一連のスパン内sdk\_invocation\_idに表示される は、ログメッセージを関連付けSDKるために によって生成された一意の ID クライアント側です。これは、からのレスポンスで見つかったリクエスト ID とは関係ありません AWS のサービス。

## ログ記録レベルを微調整する

などの環境フィルタリングをサポートする木箱を使用する場合はtracing\_subscriber、モジュールごとにログの詳細度を制御できます。

すべてのモジュールで同じログ記録レベルを有効にできます。以下は、すべてのモジュールのログ記録レベルを trace に設定します。

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

特定のモジュールのトレースレベルのログ記録を有効にできます。次の例では、からのログのみが trace レベルで受信aws\_smithy\_runtimeされます。

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

複数のモジュールに異なるログレベルを指定するには、カンマで区切ります。次の例では、aws\_configモジュールをtraceレベルロギングに設定し、aws\_smithy\_runtimeモジュールをdebugレベルロギングに設定します。

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

次の表は、ログメッセージのフィルタリングに使用できるモジュールの一部を示しています。

Prefix	説明
aws_smithy_runtime	リクエストとレスポンスのワイヤログ記録
aws_config	認証情報のロード
aws_sigv4	リクエスト署名と正規リクエスト

ログ出力に含める必要があるモジュールを特定する 1 つの方法は、まずすべてをログに記録し、次に必要な情報のログ出力で木箱名を見つけることです。その後、それに応じて環境変数を設定し、プログラムを再度実行できます。

## クライアントの 1 つのオペレーション設定を上書きする

[サービスクライアントを作成する](#)と、設定はイミュータブルになり、以降のすべてのオペレーションに適用されます。この時点では設定を変更できませんが、オペレーションごとに上書きできます。

各オペレーションビルダーには、既存の設定の個々のコピーを上書きCustomizableOperationできるように、を作成するためのcustomizeメソッドがあります。元のクライアント設定は変更されません。

次の例は、2 つのオペレーションを呼び出す Amazon S3 クライアントの作成を示しています。2 番目のオペレーションは別の に送信するように上書きされます AWS リージョン。Amazon S3 のすべ

てのオブジェクト呼び出しは、変更された を使用するようAPI呼び出しが明示的に上書きされる場合を除き、 us-east-1リージョンを使用します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

前の例は Amazon S3 用ですが、概念はすべてのオペレーションで同じです。特定のオペレーションでは、 で追加のメソッドがある場合がありますCustomizableOperation。

1 回のオペレーションで を使用してインターセプターを追加する例については、customize 「」を参照してください[特定のオペレーションのみのインターセプター](#)。

## 再試行

AWS SDK for Rust には、デフォルトの再試行動作とカスタマイズ可能な設定オプションが用意されています。を呼び出すと、予期しない例外が返 AWS のサービス されることがあります。呼び出しを再試行すると、スロットリングや一時的なエラーなど、特定のタイプのエラーが成功する可能性があります。

再試行動作は、環境変数または共有 AWS config ファイルの設定を使用してグローバルに設定できます。このアプローチの詳細については、[「およびツールリファレンスガイド」の「再試行動作」](#)を参照してください。AWS SDKs また、[再試行戦略の実装に関する詳細情報と、異なる戦略を選択する方法](#)も含まれています。

または、以下のセクションに示すように、これらのオプションをコードで設定することもできます。

### デフォルトの再試行設定

すべてのサービスクライアントは、デフォルトで `RetryConfig` 構造体を通じて提供される standard 再試行戦略設定に設定されます。デフォルトでは、呼び出しは 3 回試行されます (最初の試行と 2 回の再試行)。さらに、各再試行は、再試行ストームを避けるために、短いランダムな時間遅延します。この規則は、ほとんどのユースケースに適していますが、高スループットシステムなどの特定の状況では適さない場合があります。

では、一部のタイプのエラーのみが再試行可能と見なされます SDKs。再試行可能なエラーの例は次のとおりです。

- ソケットタイムアウト
- サービス側のスロットリング
- 5XX HTTP レスポンスなどの一時的なサービスエラー

以下の例は再試行可能とは見なされません。

- パラメータがないか無効です
- 認証/セキュリティエラー
- 設定ミスの例外

standard 再試行戦略をカスタマイズするには、最大試行回数、遅延回数、バックオフ回数を設定します。

## 最大試行回数

コードの最大試行回数をカスタマイズするには、変更した を [RetryConfig](#) に指定し  
ず `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;
```

## 遅延とバックオフ

再試行が必要な場合、デフォルトの再試行戦略は後続の試行を行う前に待機します。最初の再試行の遅延は小さいですが、後の再試行では指数関数的に増加します。最大遅延量は、大きくなりすぎないように制限されます。

ランダムジッターは、すべての試行間の遅延に適用されます。ジッターは、再試行ストームを引き起こす可能性のある大規模なフリートの影響を軽減するのに役立ちます。エクスポネンシャルバックオフとジッターの詳細については、AWS アーキテクチャブログの「[エクスポネンシャルバックオフとジッター](#)」を参照してください。

コードの遅延設定をカスタマイズするには、変更した を [RetryConfig](#) に指定し  
ず `aws_config::defaults`。次のコードは、最初の再試行を最大 100 ミリ秒遅らせるように設定し、再試行間の最大時間は 5 秒です。

```
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;
```

## アダプティブ再試行モード

standard モード再試行戦略の代わりに、adaptiveモード再試行戦略は、スロットリングエラーを最小限に抑えるための理想的なリクエストレートを求める高度なアプローチです。

### Note

アダプティブ再試行は高度な再試行モードです。この戦略を使用することは、通常お勧めしません。[「およびツールリファレンスガイド」の「再試行動作」](#)を参照してください。  
AWS SDKs

アダプティブ再試行には、標準再試行のすべての機能が含まれます。これにより、スロットリングされていないリクエストと比較したスロットリングされたリクエストのレートを測定するクライアント側のレートリミッターが追加されます。また、トラフィックが安全な帯域幅内にとどまるよう制限し、スロットリングエラーが発生するのが理想的です。

レートは、変化するサービス条件やトラフィックパターンにリアルタイムで適応し、それに応じてトラフィックレートを増減する可能性があります。クリティカルなことに、レートリミッターはトラフィックの多いシナリオでの最初の試行を遅らせる可能性があります。

コードでadaptive再試行戦略を選択するには、変更した を指定します [RetryConfig](#)。

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

## タイムアウト

AWS SDK for Rust には、リクエストのタイムアウトと停止したデータストリームを管理するためのいくつかの設定が用意されています。これらは、ネットワークで予期しない遅延や障害が発生した場合にアプリケーションが最適に動作するのに役立ちます。

## API タイムアウト

リクエストの試行に時間がかかる、または完全に失敗する可能性のある一時的な問題がある場合は、アプリケーションが迅速に失敗して最適に動作できるように、タイムアウトを確認して設定することが重要です。失敗したリクエストは、によって自動的に再試行できませんSDK。個々の試行とリクエスト全体のタイムアウトを設定することをお勧めします。

SDK for Rust は、リクエストの接続を確立するためのデフォルトのタイムアウトを提供します。には、リクエストの試行またはリクエスト全体のレスポンスを受信するためのデフォルトの最大待機時間は設定SDKされていません。次のタイムアウトオプションを使用できます。

パラメータ	デフォルト値	説明
接続タイムアウト	3.1 秒	接続を確立するのを待つ最大時間。
オペレーションタイムアウト	[なし]	すべての再試行を含む、 SDK for Rust からレスポンスを受信するまでの最大待機時間。
オペレーション試行タイムアウト	[なし]	1 回のHTTP試行を待機する最大時間。その後、API呼び出しを再試行できます。
読み取りタイムアウト	[なし]	リクエストが開始されてからレスポンスの最初のバイトを読み取るまでの最大待機時間。

次の例は、カスタムタイムアウト値を持つ Amazon S3 クライアントの設定を示しています。

```
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);
```



オペレーションタイムアウトと試行タイムアウトの両方を一緒に使用すると、再試行のすべての試行に費やされる合計時間にハード制限が設定されます。また、スローHTTPリクエストでフェイルファストするように個々のリクエストを設定します。

すべてのオペレーションに対してサービスクライアントでこれらのタイムアウト値を設定する代わりに、[単一のリクエストに対して設定または上書き](#)できます。

### ⚠ Important

SDK for Rust がレスポンスを返した後に消費されるストリーミングデータには、オペレーションタイムアウトと試行タイムアウトは適用されません。例えば、レスポンス `ByteStream` のメンバーからのデータを消費しても、オペレーションのタイムアウトは発生しません。

## ストリーム保護の停止

SDK for Rust は、停止したストリームの検出に関連する別の形式のタイムアウトを提供します。停止したストリームは、設定された猶予期間を超えてデータを生成しないアップロードストリームまたはダウンロードストリームです。これにより、アプリケーションが無期限にハングし、進行することがなくなります。

ストリームの保護が停止すると、ストリームが許容期間を超えてアイドル状態になるとエラーが返されます。

デフォルトでは、SDK for Rust はアップロードとダウンロードの両方で停止ストリームの保護を有効にし、猶予期間が 20 秒の、少なくとも 1 バイト/秒のアクティビティを探します。

次の例は、アップロード保護を無効にし、アクティビティがない猶予期間を 10 秒に変更 `StalledStreamProtectionConfig` するカスタマイズされた `Client` を示しています。

```
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**

ストリーム保護の停止は、高度な設定オプションです。これらの値は、アプリケーションがより厳密なパフォーマンスを必要とする場合や、他の問題を引き起こしている場合にのみ変更することをお勧めします。

## 停止したストリームの保護を無効にする

次の例は、停止したストリーム保護を完全に無効にする方法を示しています。

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

# を使用する AWS SDK for Rust

を使用して AWS サービス AWS SDK for Rust を操作する一般的な方法と推奨される方法について説明します。

トピック

- [リクエストを発行する](#)
- [ベストプラクティス](#)
- [同時実行](#)
- [エラー処理](#)
- [Lambda 関数を作成する](#)
- [ページ分割](#)
- [署名付き を作成する URLs](#)
- [ユニットテスト](#)
- [ウェイター](#)

## リクエストを発行する

にリクエストを行うには AWS のサービス、まず [サービスクライアントを作成](#) する必要があります。AWS のサービス コードが使用するたびに、独自の木箱とそれを操作する Client ための独自の木箱があります。

は、サービスによって Client 公開される API オペレーションごとに 1 つのメソッドを公開します。これらの各メソッドの戻り値は「fluent builder」であり、ビルダー形式の関数のコールチェーンによって異なる入力が追加され API ます。サービスの メソッドを呼び出した後、`send()` を呼び出して、出力が成功 [Future](#) する または を取得します `SdkError`。 `SdkError` の詳細については、「[エラー処理](#)」をご参照ください。

次の例は、Amazon S3 を使用して にバケットを作成する基本的なオペレーションを示しています us-west-2 AWS リージョン。

```
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;
```

各サービスクレイトには、次のようなAPI入力に使用される追加のモジュールがあります。

- `types` モジュールには、より複雑な構造化情報を提供する構造体または列挙型があります。
- `primitives` モジュールには、日時やバイナリ BLOB などのデータを表すためのよりシンプルなタイプがあります。

より詳細な木箱の組織と情報については、サービス木箱の[APIリファレンスドキュメント](#)を参照してください。例えば、Amazon Simple Storage Service `aws-sdk-s3` の木箱には複数の[モジュール](#)があります。そのうちの2つは、次のとおりです。

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

## ベストプラクティス

以下は、を使用するためのベストプラクティスです AWS SDK for Rust。

### 可能な場合はSDKクライアントを再利用する

SDK クライアントの構築方法によっては、新しいクライアントを作成すると、各クライアントが独自のHTTP接続プール、ID キャッシュなどを維持する場合があります。コストのかかるリソース作成のオーバーヘッドを避けるため`SdkConfig`、クライアントを共有するか、少なくとも共有することをお勧めします。すべてのSDKクライアントは、単一のアトミック参照数の更新`Clone`として を実装します。

## API タイムアウトを設定する

は、接続タイムアウトやソケットタイムアウトなど、一部のタイムアウトオプションのデフォルト値SDKを提供しますが、API呼び出しタイムアウトや個々のAPI呼び出し試行については提供しません。個々の試行とリクエスト全体のタイムアウトを設定することをお勧めします。これにより、リクエストの完了までの時間が長くなるような一時的な問題や、致命的なネットワーク上の問題が発生した場合に、アプリケーションが最適な方法で迅速に失敗するようになります。

オペレーションタイムアウトの設定の詳細については、「」を参照してください [タイムアウト](#)。

## 同時実行

は同時実行制御を提供し AWS SDK for Rust ませんが、ユーザーは独自の を実装するための多くのオプションがあります。

## 用語

この主題に関連する用語は混乱しやすく、いくつかの用語は、もともと別々の概念を表していましたが、シノニムになっています。このガイドでは、以下を定義します。

- **タスク:** プログラムが完了するまで実行する、または完了するまで実行を試みる「作業単位」。
- **シーケンシャルコンピューティング:** 複数のタスクが順番に実行される場合。
- **同時コンピューティング:** 複数のタスクが重複する期間で実行される場合。
- **同時実行数:** コンピュータが複数のタスクを任意の順序で完了する機能。
- **マルチタスク:** コンピュータが複数のタスクを同時に実行できる機能。
- **レース条件:** タスクの開始時期やタスクの処理にかかる時間に基づいてプログラムの動作が変わる場合。
- **競合:** 共有リソースへのアクセスの競合。2 つ以上のタスクが同時にリソースにアクセスする場合、そのリソースは「競合中」になります。
- **デッドロック:** これ以上進行できない状態。これは通常、2 つのタスクが互いのリソースを取得したいが、どちらのタスクも他方のリソースが使用可能になるまでリソースを解放しないために発生します。デッドロックにより、プログラムが部分的または完全に応答しなくなります。

## 簡単な例

最初の例は、シーケンシャルプログラムです。後の例では、同時実行技術を使用してこのコードを変更します。後の例では、同じ`build_client_and_list_objects_to_download()`メソッドを再利用し、内で変更を行います`main()`。

次のタスク例では、Amazon Simple Storage Service バケット内のすべてのファイルをダウンロードします。

1. まず、すべてのファイルを一覧表示します。キーをリストに保存します。
2. リストを繰り返し、各ファイルを順番にダウンロードする

```
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

これらの例では、エラーは処理されず、サンプルバケットにはファイルパスのようなキーを持つオブジェクトがないと仮定します。したがって、ネストされたディレクトリの作成については取り上げません。

最新のコンピュータのアーキテクチャにより、このプログラムを書き換えて、はるかに効率的にすることが可能です。これについては、後の例で説明しますが、まず、いくつかの概念を学習しましょう。

## 所有権とミュータビリティ

Rust の各値には 1 つの所有者があります。所有者が範囲外になると、所有者が所有するすべての値も削除されます。所有者は、値への 1 つ以上のイミュータブルな参照または 1 つのミュータブルな参照のいずれかを指定できます。Rust コンパイラは、参照がその所有者を上回らないようにする責任があります。

複数のタスクが同じリソースに可変的にアクセスする必要がある場合は、追加の計画と設計が必要です。シーケンシャルコンピューティングでは、各タスクは順番に実行されるため、競合することなく同じリソースに可変的にアクセスできます。ただし、同時コンピューティングでは、タスクは任意の順序で同時に実行できます。したがって、複数の変更可能な参照が不可能である (または、変更可能な参照が発生した場合は少なくともクラッシュする) ことをコンパイラに証明するために、さらに多くのことを実行する必要があります。

Rust 標準ライブラリには、これを実現するのに役立つ多くのツールが用意されています。これらのトピックの詳細については、[「Rust プログラミング言語」の「変数とミュータビリティ」](#)および[「所有権について」](#)を参照してください。

## その他の用語

以下に、「同期オブジェクト」のリストを示します。これらは、コンパイラが同時実行プログラムによって所有権のルールが破られないことを確信するために必要なツールです。

### 標準ライブラリ同期オブジェクト :

- **Arc**: A トーム R 推論 C の冪定ポインタ。データが でラップされるとArc、特定の所有者が値を早期に削除することを心配することなく、自由に共有できます。この意味では、値の所有権は「共有」になります。内の値はミュータブルArcにすることはできませんが、内部ミュータブルにすることができます。
- **より危険**: 複数のスレッドが相互にプログラム内のポイントに到達するのを待ってから、すべて一緒に実行を続行します。
- **Condvar**: イベントの発生を待っている間にスレッドをブロックする機能を提供する Condition Variable。
- **Mutex**: 一度に最大 1 つのスレッドが一部のデータにアクセスできるようにする相互除外メカニズム。一般的に、Mutexロックはコード内の 1 つの .await ポイントにまたがって保持しないでください。

### Tokio 同期オブジェクト :

は async-runtime-agnostic AWS SDKsを目的としていますが、特定のケースではtokio同期オブジェクトを使用することをお勧めします。

- **Mutex**: 標準ライブラリの に似ていますがMutex、コストがわずかに高くなります。標準のとは異なりMutex、これはコード内の 1 つの .await ポイントにまたがって保持できます。
- **Semaphore**: 複数のタスクによって共通リソースへのアクセスを制御するために使用される変数。

## 例をより効率的に書き換える (シングルスレッド同時実行)

次の変更された例では、futures\_util::future::join\_allを使用してALLget\_objectリクエストを同時に実行します。

```
#[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;
}
```

これは、同時実行のメリットを享受する最も簡単な方法ですが、一見して明らかでない問題がいくつかあります。

1. すべてのリクエスト入力を同時に作成します。すべての`get_object`リクエスト入力を保持するのに十分なメモリがない場合、`out-of-memory`「」割り当てエラーが発生します。
2. すべての未来を同時に作成して待ちます。Amazon S3は、一度にダウンロードしようとする回数が多すぎるとリクエストを調整します。

これらの問題の両方を修正するには、一度に送信するリクエストの量を制限する必要があります。これを行うには、[tokioセマフォ](#)を使用します。

```
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;
}
```

リクエストの作成を `async` ブロックに移動することで、潜在的なメモリ使用量の問題を修正しました。これにより、リクエストは送信するまで作成されません。

#### Note

メモリがある場合は、すべてのリクエスト入力を一度に作成し、送信の準備ができるまでメモリに保持する方が効率的です。これを試すには、リクエスト入力の作成を `async` ブロックの外部に移動します。

また、転送中のリクエストを `CONCURRENCY_LIMIT` に制限することで、一度に送信するリクエストが多すぎる問題を修正しました。

**Note**

の適切な値はCONCURRENCY\_LIMIT、プロジェクトごとに異なります。独自のリクエストを作成して送信する場合は、スロットリングエラーが発生しないように、できるだけ高く設定してください。同時実行数の制限は、サービスが送信する成功レスポンスとスロットリングレスポンスの比率に基づいて動的に更新できますが、その複雑さからこのガイドの対象外です。

## 例をより効率的に書き換える (マルチスレッド同時実行)

前の2つの例では、リクエストを同時に実行しました。これは同期的に実行するよりも効率的ですが、マルチスレッドを使用することで、より効率的になります。でこれを行うにはtokio、それらを個別のタスクとしてスポンする必要があります。

**Note**

この例では、マルチスレッドtokioランタイムを使用する必要があります。このランタイムはrt-multi-thread機能の背後でゲートされます。また、もちろん、マルチコアマシンでプログラムを実行する必要があります。

```
// 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;
}
```

作業をタスクに分割するのは複雑になる場合があります。通常、I/O (入力/出力) の実行はブロックされます。ランタイムは、長時間実行されるタスクのニーズと短時間実行されるタスクのニーズのバランスをとるのに苦労する可能性があります。どのランタイムを選択しても、作業をタスクに分割する最も効率的な方法に関する推奨事項を必ずお読みください。tokio ランタイムの推奨事項については、[「モジュールtokio::task」](#)を参照してください。

## マルチスレッドアプリケーションのデバッグ

同時に実行されるタスクは、任意の順序で実行できます。そのため、同時実行プログラムのログを読み取ることが非常に難しい場合があります。SDK for Rust では、tracingログ記録システムを使用することをお勧めします。実行中であっても、特定のタスクでログをグループ化できます。ガイダンスについては、[「AWS SDK for Rust コードのログ記録を有効にする」](#)を参照してください。

ロックアップされたタスクを識別するための非常に便利なツールは [tokio-console](#) です。これは [tokio-console](#)、非同期 Rust プログラムの診断およびデバッグツールです。プログラムを計測して実行し、tokio-consoleアプリを実行すると、プログラムが実行中のタスクのライブビューが表示されます。このビューには、タスクが共有リソースの取得を待つのにかかった時間や、ポーリングされた時間などの有用な情報が含まれます。

## エラー処理

を使用して高品質のアプリケーションを構築するには、がエラーをいつどのように AWS SDK for Rust 返すかを理解することが重要です SDK。以下のセクションでは、で発生する可能性のあるさまざまなエラー SDK と、それらを適切に処理する方法について説明します。

すべてのオペレーションは、エラー Result タイプが に設定されたタイプを返します `SdkError<E, R = HttpResponse>`。SdkError は、バリエントと呼ばれるいくつかの可能なタイプを持つ列挙型です。

## サービスエラー

エラーの最も一般的なタイプは です `SdkError::ServiceError`。このエラーは、からのエラーレスポンスを表します AWS のサービス。例えば、存在しないオブジェクトを Amazon S3 から取得しようとする、Amazon S3 はエラーレスポンスを返します。

が発生する `SdkError::ServiceError` と、リクエストは に正常に送信されました AWS のサービスが、処理できなかったことを意味します。これは、リクエストのパラメータに含まれるエラーまたはサービス側の問題が原因です。

エラーレスポンスの詳細は、エラーバリエントに含まれています。次の例は、基盤となる `ServiceError` バリエントを簡単に取得し、さまざまなエラーケースを処理する方法を示しています。

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

## エラーメタデータ

すべてのサービスエラーには、サービス固有の特性をインポートすることでアクセスできる追加のメタデータがあります。

- `<service>::error::ProvideErrorMetadata` 特性は、サービスから返される利用可能な基盤となる生のエラーコードとエラーメッセージへのアクセスを提供します。
  - Amazon S3 の場合、この特性は [です `aws\_sdk\_s3::error::ProvideErrorMetadata`](#)。

サービスエラーのトラブルシューティングに役立つ情報を取得することもできます。

- `<service>::operation::RequestId` 特性は、サービスによって生成された一意の AWS リクエスト ID を取得するための拡張メソッドを追加します。
  - Amazon S3 の場合、この特性は [です `aws\_sdk\_s3::operation::RequestId`](#)。
- トrait は `<service>::operation::RequestIdExt`、`extended_request_id()` メソッドを追加して、拡張リクエスト ID を追加します。
  - 一部のサービスでのみサポートされています。
  - Amazon S3 の場合、この特性は [です `aws\_sdk\_s3::operation::RequestIdExt`](#)。

## を使用した詳細なエラー印刷 `DisplayErrorContext`

のエラー SDK は通常、次のような一連の障害の結果です。

1. コネクタがエラーを返したため、リクエストのディスパッチに失敗しました。
2. 認証情報プロバイダーがエラーを返したため、コネクタはエラーを返しました。
3. 認証情報プロバイダーが サービスを呼び出し、そのサービスがエラーを返したため、エラーが返されました。
4. 認証情報リクエストに正しい認可がないため、サービスからエラーが返されました。

デフォルトでは、このエラーの表示は「パッチ失敗」のみを出力します。これには、エラーのトラブルシューティングに役立つ詳細がありません。SDK for Rust には、というシンプルなエラーレポーターが用意されています `DisplayErrorContext`。

- `<service>::error::DisplayErrorContext` 構造体は、完全なエラーコンテキストを出力する機能を追加します。
- Amazon S3 の場合、この構造体は [です `aws\_sdk\_s3::error::DisplayErrorContext`](#)。

エラーをラップして表示し、出力すると、は次のようなより詳細なメッセージ `DisplayErrorContext` を提供します。

```
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") },
```

```

    H0("application/json") },
    { _private: H0("114") },
    HeaderValue { _private: H0("RequestId") },
    HeaderValue { _private: H0("x-amzn-RequestId") },
    H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") },
    H0("AWS SSO") },
    { _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 関数を作成する

を使用した AWS Lambda 関数の開発に関する詳細なドキュメントについては AWS SDK for Rust、AWS Lambda デベロッパーガイドの「[Rust を使用した Lambda 関数の構築](#)」を参照してください。このドキュメントでは、以下を使用する方法について説明します。

- コア機能用の Rust Lambda ランタイムクライアントクレート、[aws-lambda-rust-runtime](#)。
- [Cargo Lambda を使用して Rust 関数バイナリを Lambda にデプロイするための推奨コマンドラインツール](#)。

AWS Lambda デベロッパーガイドのガイド付き例に加えて、[の AWS SDK コード例リポジトリ](#)で Lambda 計算ツールの例も使用できます GitHub。

## ページ分割

多くの AWS オペレーションは、ペイロードが大きすぎて 1 回のレスポンスで返せない場合に、切り捨てられた結果を返します。代わりに、サービスはデータの一部とトークンを返し、次の項目のセットを取得します。このパターンはページ分割と呼ばれます。

AWS SDK for Rust には、結果を自動的にページ分割するために使用できる `into_paginator` オペレーションビルダーの拡張メソッドが含まれています。結果を処理するコードを作成するだけで済みます。すべてのページ分割オペレーションビルダーには、結果をページ分割 `PaginationStream<Item>` する を公開する `into_paginator()` メソッドがあります。

- Amazon S3 では、その一例が [aws\\_sdk\\_s3::operation::list\\_objects\\_v2::builders::ListObjectsV2FluentBuilder](#) で

次の例では、Amazon Simple Storage Service を使用しています。ただし、概念は、ページ分割された が 1 つ以上あるサービスでも同じです APIs。

次のコード例は、[try\\_collect\(\)](#) メソッドを使用してページ分割されたすべての結果を に収集する最も簡単な例を示しています 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<_>>();
```

ページングをより詳細に制御し、すべてを一度にメモリに取り込まないようにしたい場合があります。次の例では、それ以上なくなるまで Amazon S3 バケット内のオブジェクトを反復処理します。

```
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);
    }
}
```

## 署名付き を作成する URLs

一部のオペレーションの AWS API リクエストに事前署名して、別の発信者が自分の認証情報を提示せずに後でリクエストを使用できるようにすることができます。

たとえば、Jane が Amazon Simple Storage Service (Amazon S3) オブジェクトにアクセスできるが、Alejandro とオブジェクトアクセスを一時的に共有したいとします。Jane は、署名付き GetObject リクエストを生成して Alejandro と共有できるため、Jane の認証情報へのアクセスや独自の認証情報の取得を必要とせずにオブジェクトをダウンロードできます。署名付き が使用する認証情報 URL は、ジェーンが を生成した AWS ユーザーであるため、ジェーン の です URL。

Amazon S3 URLs での署名付き の詳細については、Amazon Simple Storage Service ユーザーガイドの「[署名付き の使用URLs](#)」を参照してください。Amazon S3

## 署名の基本

AWS SDK for Rust は、署名付き リクエストの取得に使用できるオペレーション fluent-builder の `presigned()` メソッドを提供します。

次の例では、Amazon S3 の署名付き GetObject リクエストを作成します。リクエストは作成後 5 分間有効です。

```
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()` メソッドは を返します `Result<PresignedRequest, SdkError<E, R>>`。

返される `PresignedRequest` には、メソッド、および任意のヘッダーを含む URI HTTP リクエストのコンポーネントで取得するメソッドが含まれています。リクエストを有効にするには、これらすべてが存在する場合はサービスに送信する必要があります。ただし、多くの署名付きリクエストは URI だけで表すことができます。

## および POSTPUTリクエストの事前署名

事前署名可能な多くのオペレーションはのみを必要としURL、HTTPGETリクエストとして送信する必要があります。ただし、一部のオペレーションは本文を取り、場合によってはヘッダーとともにHTTPPOSTまたはHTTPPUTリクエストとして送信する必要があります。これらのリクエストの事前署名はGETリクエストの事前署名と同じですが、署名付きリクエストの呼び出しはより複雑です。

以下は、Amazon S3 PutObjectリクエストに事前署名し、選択した HTTPクライアントを使用して送信[http::request::Request](http://request::Request)できる に変換する例です。

`into_http_1x_request()` メソッドを使用するには、Cargo.tomlファイルのaws-sdk-s3クレートに `http-1x`機能を追加します。

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

ソースファイル：

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

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

## スタンドアロン署名者

### Note

これは高度なユースケースです。ほとんどのユーザーには必要でも推奨でもありません。

SDK for Rust コンテキストの外部で署名付きリクエストを作成する必要があるユースケースがいくつかあります。そのため、[aws-sigv4](#)とは別に木箱を使用できますSDK。

以下は、基本的な要素を示す例です。詳細については、木箱のドキュメントを参照してください。

aws-sigv4 および http クレートを Cargo.toml ファイルに追加します。

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

ソースファイル :

```
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);
```

## ユニットテスト

AWS SDK for Rust プロジェクトにユニットテストを実装する方法は多数ありますが、いくつかの方法をお勧めします。

- [mockall](#) クレート [automock](#) からを使用して、テストを作成および実行します。
- AWS Smithy ランタイムの [StaticReplayClient](#) を使用して、通常は が使用する標準HTTPクライアントの代わりに使用できるフェイクHTTPクライアントを作成します AWS のサービス。このクライアントは、ネットワーク経由でサービスと通信するのではなく、指定したHTTPレスポンスを返します。これにより、テストはテスト目的で既知のデータを取得します。

## モックオールを使用してモックを自動的に生成する

[mockall](#) クレート [automock](#) の一般的な を使用して、テストに必要なモック実装の大部分を自動的に生成できます。

この例では、 というカスタムメソッドをテストします `determine_prefix_file_size()`。このメソッドは、Amazon S3 を呼び出すカスタム `list_objects()` ラッパーメソッドを呼び出します。Amazon S3 をモックすることで `list_objects()`、Amazon S3 に実際に連絡することなく `determine_prefix_file_size()` メソッドをテストできます。

1. プロジェクトディレクトリのコマンドプロンプトで、依存関係として [mockall](#) 木箱を追加します。

```
$ cargo add mockall
```

これにより、Cargo.toml ファイルの [dependencies] セクションに木箱が追加されます。

2. `mockall` 木箱から `automock` モジュールを含めます。

テスト AWS のサービス する に関連する他のライブラリ、この場合は Amazon S3 も含めません。

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

```
use mockall::automock;

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

- 次に、アプリケーションの Amazon S3 ラッパー構造の 2 つの実装のうちどれを使用するかを決定するコードを追加します。
  - ネットワーク経由で Amazon S3 にアクセスするように記述された実際のもの。
  - によって生成されたモック実装 `mockall`。

この例では、選択した `list_objects_v2` という名前が付けられています。選択は、`test` 属性に基づいて条件付きです。

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

- `S3Impl` 構造体は、実際に `list_objects_v2` リクエストを送信する Amazon S3 ラッパー構造の実装です AWS。
  - テストが有効になっている場合、リクエストはモックに送信され、モックに送信されないため、このコードは使用されません AWS。 `dead_code` 属性は、`S3Impl` タイプが使用されていない場合に問題を報告しないように linter に指示します。
  - 条件付き `test` は、テストが有効になっている場合、`automock` 属性を設定する必要があります `#[cfg_attr(test, automock)]` ことを示します。これは、`list_objects_v2` という名前のモックを生成する `mockall` ように `list_objects_v2` に指示 `S3Impl` します `MockS3Impl`。
  - この例では、`list_objects_v2()` メソッドはモックする呼び出しです。 `automock` は自動的に `expect_list_objects_v2()` メソッドを作成します。

```
#[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. という名前のモジュールでテスト関数を作成しますtest。

- 条件付きは、test 属性が の場合、 がテストモジュールを構築mockallする必要がある#[cfg(test)]ことを示しますtrue。

```

#[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);
}
}

```

- 各テストでは `let mut mock = MockS3Impl::default();`、を使用しての `mock` インスタンスを作成します `MockS3Impl`。
  - モックの `expect_list_objects()` メソッド (によって自動的に作成された `automock`) を使用して、コード内の別の場所で `list_objects()` メソッドが使用されたときの期待される結果を設定します。
  - 期待値が確立されると、これらを使用して を呼び出して関数をテストします `determine_prefix_file_size()`。返された値は、アサーションを使用して正しいことを確認するためにチェックされます。
6. `determine_prefix_file_size()` 関数は Amazon S3 ラッパーを使用してプレフィックスファイルのサイズを取得します。

```

#[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)
}
```

タイプS3は、HTTPリクエストを行うMockS3Implときに S3Implと の両方をサポートするように、ラップされた SDK for Rust 関数を呼び出すために使用されます。テストが有効になっている場合、によって自動的に生成されたモックはテスト失敗mockallを報告します。

[これらの例の完全なコードは、](#) で確認できます GitHub。

## 静的再生を使用してHTTPトラフィックをシミュレートする

aws-smithy-runtime クレートには、 というテストユーティリティクラスが含まれていませ [StaticReplayClient](#)。このHTTPクライアントクラスは、AWS のサービス オブジェクトの作成時にデフォルトのHTTPクライアントの代わりに指定できます。

を初期化するときはStaticReplayClient、HTTPリクエストとレスポンスのペアのリストをReplayEventオブジェクトとして指定します。テストの実行中に、各HTTPリクエストが記録され、クライアントはイベントリストReplayEventの次の にある次のHTTPレスポンスをHTTPクライアントのレスポンスとして返します。これにより、ネットワーク接続なしで既知のデータを使用してテストを実行できます。

### 静的再生の使用

静的再生を使用するには、ラッパーを使用する必要はありません。代わりに、テストで使用するデータに対する実際のネットワークトラフィックがどのように見えるかを判断し、 が AWS のサービスクライアントからリクエストSDKを発行するたびにStaticReplayClient、そのトラフィックデータを に提供します。

#### Note

予想されるネットワークトラフィックを収集するには、AWS CLI や多数のネットワークトラフィックアナライザー、パケットスニッファーツールなど、いくつかの方法があります。

- 予想されるHTTPリクエストと返されるレスポンスを指定するReplayEventオブジェクトのリストを作成します。
- 前のステップで作成したHTTPトランザクションリストStaticReplayClientを使用してを作成します。
- AWS クライアントの設定オブジェクトを作成し、をConfigオブジェクトのStaticReplayClientとして指定しますhttp\_client。
- 前のステップで作成した設定を使用して、AWS のサービス クライアントオブジェクトを作成します。
- を使用するように設定されたサービスオブジェクトを使用して、テストするオペレーションを実行しますStaticReplayClient。が APIリクエストSDKを送信するたびに AWS、リスト内の次のレスポンスが使用されます。

### Note

送信されたリクエストがReplayEventオブジェクトのベクトルのリクエストと一致しない場合でも、リスト内の次のレスポンスは常に返されます。

- 必要なリクエストがすべて実行されたら、StaticReplayClient.assert\_requests\_match()関数を呼び出して、によって送信されたリクエストがReplayEventオブジェクトのリストのリクエストSDKと一致することを確認します。

## 例

前の例で同じdetermine\_prefix\_file\_size()関数のテストを見てみましょうが、モックの代わりに静的再生を使用します。

1. プロジェクトディレクトリのコマンドプロンプトで、依存関係として[aws-smithy-runtime](#)木箱を追加します。

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

これにより、Cargo.toml ファイルの [dependencies]セクションに木箱が追加されます。

2. ソースファイルに、必要なaws\_smithy\_runtimeタイプを含めます。

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

```
use aws_smithy_types::body::SdkBody;
```

3. テストは、テスト中に実行する必要がある各HTTPトランザクションを表すReplayEvent構造の作成から始まります。各イベントには、HTTPリクエストオブジェクトと、がAWSのサービス通常応答する情報を表すHTTPレスポンスオブジェクトが含まれます。これらのイベントは、への呼び出しに渡されますStaticReplayClient::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]);
```

結果は に保存されますreplay\_client。これは、HTTPクライアントの設定で指定することで、SDK for Rust で使用できる クライアントを表します。

4. Amazon S3 クライアントを作成するには、クライアントクラスの from\_conf()関数を呼び出して、設定オブジェクトを使用してクライアントを作成します。

```
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(),
);
```

設定オブジェクトはビルダーの `http_client()` メソッドを使用して指定され、認証情報は `credentials_provider()` メソッドを使用して指定されます。認証情報は、偽の認証情報構造を返す `make_s3_test_credentials()` という関数を使用して作成されます。

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

これらの認証情報は実際には送信されないため、有効である必要はありません AWS。

5. テストが必要な 関数を呼び出してテストを実行します。この例では、その関数の名前は `determine_prefix_file_size()` です。最初のパラメータは、リクエストに使用する Amazon S3 クライアントオブジェクトです。したがって、を使用して作成されたクライアントを指定 `StaticReplayClient` し、リクエストがネットワーク経由で送信されるのではなく、そのクライアントによって処理されるようにします。

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

assert_eq!(19, size);

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

の呼び出し `determine_prefix_file_size()` が完了すると、アサートを使用して、返された値が想定値と一致することを確認します。次に、 `StaticReplayClient` メソッド `assert_requests_match()` 関数が呼び出されます。この関数は、記録された HTTP リクエ

ストをスキャンし、再生クライアントの作成時に提供されたReplayEventオブジェクトの配列で指定されたリクエストとすべて一致することを確認します。

[これらの例の完全なコードは、](#) で確認できます GitHub。

## ウェイター

ウェーターは、目的の状態に達するまで、またはリソースが目的の状態に移行しないと判断されるまで、リソースをポーリングするために使用されるクライアント側の抽象化です。これは、Amazon Simple Storage Service などの結果整合性のあるサービスや、Amazon Elastic Compute Cloud などのリソースを非同期的に作成するサービスを使用する場合の一般的なタスクです。リソースのステータスを継続的にポーリングするロジックを記述すると、面倒でエラーが発生しやすくなります。ウェーターの目標は、この責任をカスタマーコードから に移動することです。この は AWS SDK for Rust、AWS オペレーションのタイミングの側面に関する深い知識を持っています。

AWS のサービス ウェイターをサポートするには、 `<service>::waiters` モジュールが含まれます。

- `<service>::client::Waiters` トレイトは、クライアントのウェーターメソッドを提供します。メソッドは Client 構造体に実装されます。すべてのウェーターメソッドは、 の標準の命名規則に従います。 `wait_until_<Condition>`
- Amazon S3 の場合、この特性は です [aws\\_sdk\\_s3::client::Waiters](#)。

次の例では、Amazon S3 を使用しています。ただし、概念は、1 つ以上のウェーター AWS のサービスが定義されている でも同じです。

次のコード例は、作成後にバケットが存在するのを待つポーリングロジックを記述する代わりに、ウェーター関数を使用する方法を示しています。

```
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

各待機メソッド `Result<FinalPoll<...>, WaiterError<...>>` は、目的の条件またはエラーに達した場合の最終応答時に取得するために使用できる `wait` を返します。詳細については、Rust APIドキュメント [WaiterError](#) の [FinalPoll](#) 「」 および 「」 を参照してください。



# SDK for Rust コード例

このトピックのコード例は、で for Rust を使用する AWS SDK方法を示しています AWS。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

一部のサービスには、サービスに固有のライブラリまたは関数を活用する方法を示す追加のカテゴリの例が含まれています。

## サービス

- [API SDK for Rust を使用したゲートウェイの例](#)
- [API SDK for Rust を使用したゲートウェイ管理APIの例](#)
- [SDK for Rust を使用した Application Auto Scaling の例](#)
- [SDK for Rust を使用した Aurora の例](#)
- [SDK for Rust を使用した Auto Scaling の例](#)
- [SDK for Rust を使用した Amazon Bedrock ランタイムの例](#)
- [SDK for Rust を使用した Amazon Cognito ID プロバイダーの例](#)
- [SDK for Rust を使用した Amazon Cognito Sync の例](#)
- [SDK for Rust を使用した Firehose の例](#)
- [SDK for Rust を使用した Amazon DocumentDB の例](#)
- [SDK for Rust を使用した DynamoDB の例](#)
- [SDK for Rust を使用した Amazon EBSの例](#)
- [SDK for Rust を使用した Amazon EC2の例](#)
- [SDK for Rust を使用した Amazon ECRの例](#)
- [SDK for Rust を使用した Amazon ECSの例](#)

- [SDK for Rust を使用した Amazon EKS の例](#)
- [AWS Glue SDK for Rust を使用した の例](#)
- [IAM SDK for Rust を使用した の例](#)
- [AWS IoT SDK for Rust を使用した の例](#)
- [SDK for Rust を使用した Kinesis の例](#)
- [AWS KMS SDK for Rust を使用した の例](#)
- [SDK for Rust を使用した Lambda の例](#)
- [MediaLive SDK for Rust を使用した の例](#)
- [MediaPackage SDK for Rust を使用した の例](#)
- [SDK for Rust を使用した Amazon Polly の例](#)
- [QLDB SDK for Rust を使用した の例](#)
- [SDK for Rust を使用した Amazon RDS の例](#)
- [SDK for Rust を使用した Amazon RDS Data Service の例](#)
- [SDK for Rust を使用した Amazon Rekognition の例](#)
- [SDK for Rust を使用した Route 53 の例](#)
- [SDK for Rust を使用した Amazon S3 の例](#)
- [SageMaker SDK for Rust を使用した AI の例](#)
- [SDK for Rust を使用した Secrets Manager の例](#)
- [SDK for Rust SES API を使用した Amazon v2 の例](#)
- [SDK for Rust を使用した Amazon SNS の例](#)
- [SDK for Rust を使用した Amazon SQS の例](#)
- [AWS STS SDK for Rust を使用した の例](#)
- [SDK for Rust を使用した Systems Manager の例](#)
- [SDK for Rust を使用した Amazon Transcribe の例](#)

## API SDK for Rust を使用したゲートウェイの例

次のコード例は、for Rust with API Gateway を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1つのサービス内から、または他のAWSのサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS コミュニティへの貢献は、複数のチームが作成し、維持している例です AWS。フィードバックを提供するには、リンクされたリポジトリで提供されているメカニズムを使用します。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)
- [シナリオ](#)
- [AWS コミュニティへの貢献](#)

## アクション

### GetRestApis

次の例は、GetRestApis を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

リージョンRESTAPIsの Amazon API Gateway を表示します。

```
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 詳細については、AWS SDK Rust API リファレンスの [GetRestApis](#) 「」の「」を参照してください。

## シナリオ

### サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

#### SDK Rust の

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードとセットアップおよび実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例のソースについて詳しくは、[AWS コミュニティ](#) でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition

- Amazon S3
- Amazon SNS

## AWS コミュニティへの貢献

### サーバーレスアプリケーションの構築とテスト

次のコード例は、Lambda と DynamoDB で API Gateway を使用してサーバーレスアプリケーションを構築およびテストする方法を示しています。

### SDK Rust の

Rust を使用して、Lambda と DynamoDB を備えた API ゲートウェイで構成されるサーバーレスアプリケーションを構築してテストする方法を示しますSDK。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください[GitHub](#)。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda

## API SDK for Rust を使用したゲートウェイ管理APIの例

次のコード例は、APIゲートウェイ管理 で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示していますAPI。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### トピック

- [アクション](#)

# アクション

## PostToConnection

次の例は、PostToConnection を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、[PostToConnection](#) 「」のAWS SDK「Rust APIリファレンス」を参照してください。

## SDK for Rust を使用した Application Auto Scaling の例

次のコード例は、アプリケーションの Auto Scaling で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

### アクション

#### DescribeScalingPolicies

次の例は、DescribeScalingPolicies を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeScalingPolicies](#) 「」の「」を参照してください。

## SDK for Rust を使用した Aurora の例

次のコード例は、Aurora で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### 開始方法

Hello Aurora

次のコード例は、Aurora の使用を開始する方法を示しています。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。



```
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 詳細については、AWS SDK「Rust APIリファレンス」の「の [DescribeDBClusters](#)」を参照してください。

## トピック

- [基本](#)
- [アクション](#)

## 基本

### 基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- カスタム Aurora DB クラスターパラメータグループを作成し、パラメータ値を設定します。
- パラメータグループを使用する DB クラスターを作成する
- データベースを含む DB インスタンスを作成します。
- DB クラスターのスナップショットを作成して、リソースをクリーンアップします。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

Aurora シナリオのシナリオ固有の関数を含むライブラリ。

```
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 クライアントラッパーの周囲にオートモックを使用してライブラリをテストします。

```
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");
}
```

ユーザーが一部の決定を下せるように `inquirer` を使用してシナリオを最初から最後まで実行するためのバイナリ。

```
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!("{}", 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
        }
    }
}
}
```

テストの自動モックを許可する Amazon RDSサービスのラッパー。

```
use aws_sdk_rds::{
    error::SdkError,
```



```

operation::{
    create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
    create_db_cluster_parameter_group::{CreateDBClusterParameterGroupError,
    create_db_cluster_parameter_group::{CreateDbClusterParameterGroupOutput,
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
}
}
```

このシナリオで使用した依存関係を含む Cargo.toml。

```
[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 詳細については、AWS SDK Rust API リファレンスの以下のトピックを参照してください。
  - [CreateDBCluster](#)
  - [CreateDBClusterParameterGroup](#)
  - [CreateDBClusterスナップショット](#)
  - [CreateDBInstance](#)
  - [DeleteDBCluster](#)
  - [DeleteDBClusterParameterGroup](#)
  - [DeleteDBInstance](#)
  - [DescribeDBClusterParameterGroups](#)
  - [DescribeDBClusterパラメータ](#)

- [DescribeDBCluster](#)スナップショット
- [DescribeDBClusters](#)
- [DescribeDBEngine](#)バージョン
- [DescribeDBInstances](#)
- [DescribeOrderableDBInstanceOptions](#)
- [ModifyDBClusterParameterGroup](#)

## アクション

### CreateDBCluster

次の例は、CreateDBCluster を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「 [」を参照してください GitHub。用例一覧を検索し、\[AWS コード例リポジトリ\]\(#\)での設定と実行の方法を確認してください。](#)

```
// 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 詳細については、AWS SDK 「Rust APIリファレンス」の「」の「[CreateDBCluster](#)」を参照してください。

## CreateDBClusterParameterGroup

次のコード例は、CreateDBClusterParameterGroup を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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 詳細については、「Rust リファレンス[createDBClusterParameterGroup](#)」の「C」を参照してください。AWS SDK API

## CreateDBClusterSnapshot

次のコード例は、CreateDBClusterSnapshot を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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");
        })
        .returning(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 詳細については、[「Rust reateDBClusterリファレンス」の「の C スナップショット」](#)を参照してください。AWS SDK API

## CreateDBInstance

次の例は、CreateDBInstance を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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_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 詳細については、AWS SDK「Rust APIリファレンス」の「」の「[CreateDBInstance](#)」を参照してください。

## DeleteDBCluster

次の例は、DeleteDBCluster を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK 「Rust APIリファレンス」の「の [DeleteDBCluster](#)」を参照してください。

## DeleteDBClusterParameterGroup

次のコード例は、DeleteDBClusterParameterGroup を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

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(
```



```
    }

    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 詳細については、AWS SDK 「Rust APIリファレンス」の「[DeleteDBClusterParameterGroup](#)」を参照してください。

## DeleteDBInstance

次の例は、DeleteDBInstance を使用する方法を説明しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK「Rust APIリファレンス」の「の [DeleteDBInstance](#)」を参照してください。

## DescribeDBClusterParameters

次の例は、DescribeDBClusterParameters を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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 詳細については、AWS SDK「Rust APIリファレンス」の「の [DescribeDBClusterパラメータ](#)」を参照してください。

## DescribeDBClusters

次のコード例は、DescribeDBClusters を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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 詳細については、AWS SDK 「Rust APIリファレンス」の「の [DescribeDBClusters](#)」を参照してください。

## DescribeDBEngineVersions

次のコード例は、DescribeDBEngineVersions を使用する方法を示しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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

#[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"
    );
}

```

```
}
```

- API 詳細については、[「Rust リファレンス」の「DescribeDBEngineバージョン」](#)を参照してください。AWS SDK API

## DescribeDBInstances

次のコード例は、DescribeDBInstances を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK「Rust APIリファレンス」の「[DescribeDBInstances](#)」を参照してください。

## DescribeOrderableDBInstanceOptions

次の例は、DescribeOrderableDBInstanceOptions を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeOrderableDBInstanceOptions](#) 「」の「」を参照してください。

## ModifyDBClusterParameterGroup

次の例は、ModifyDBClusterParameterGroup を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

// 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 詳細については、AWS SDK「Rust APIリファレンス」の「の [ModifyDBClusterParameterGroup](#)」を参照してください。

## SDK for Rust を使用した Auto Scaling の例

次のコード例は、Auto Scaling で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### 開始方法

こんにちは、Auto Scaling

次のコード例は、Auto Scaling の使用を開始する方法を示しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeAutoScalingGroups](#) 「」の「」を参照してください。

## トピック

- [基本](#)
- [アクション](#)

## 基本

### 基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- 起動テンプレートとアベイラビリティゾーンを使用して Amazon EC2 Auto Scaling グループを作成し、実行中のインスタンスに関する情報を取得します。
- Amazon CloudWatch メトリクス収集を有効にします。
- グループの希望するキャパシティを更新し、インスタンスが起動するのを待ちます。
- グループ内の最も古いインスタンスを削除します。
- ユーザーのリクエストやキャパシティの変更に応じて発生するスケーリングアクティビティを一覧表示します。
- CloudWatch メトリクスの統計を取得し、リソースをクリーンアップします。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
[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::
```

```
}

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}"),
        }

        writeln!(f, "\t\t\t\t\t Activities:");
        match &self.activities {
            Ok(activities) => {
                for activity in activities {
```

```
        writeln!(
            f,
            "\t\t- {} Progress: {}% Status: {:?} End: {:?}",
            activity.cause().unwrap_or("Unknown"),
            activity.progress.unwrap_or(-1),
            activity.status_code(),
            // activity.status_message().unwrap_or_default()
            activity.end_time(),
        )?;
    }
}
Err(e) => writeln!(f, "\t\t! {e}")?,
}

Ok(())
}
}

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

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

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, "{}")
    }
}
```

```
#[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::
```

```

        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::
```



```
        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 詳細については、AWS SDK Rust API リファレンスの以下のトピックを参照してください。
  - [CreateAutoScalingGroup](#)
  - [DeleteAutoScalingGroup](#)
  - [DescribeAutoScalingGroups](#)
  - [DescribeAutoScalingInstances](#)

- [DescribeScalingActivities](#)
- [DisableMetricsCollection](#)
- [EnableMetricsCollection](#)
- [SetDesiredCapacity](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

## アクション

### CreateAutoScalingGroup

次の例は、CreateAutoScalingGroup を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateAutoScalingGroup](#) 「」の「」を参照してください。

## DeleteAutoScalingGroup

次の例は、DeleteAutoScalingGroup を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの[DeleteAutoScalingGroup](#)「」の「」を参照してください。

## DescribeAutoScalingGroups

次のコード例は、DescribeAutoScalingGroups を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeAutoScalingGroups](#) 「」の「」を参照してください。

## DescribeAutoScalingInstances

次のコード例は、DescribeAutoScalingInstances を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust APIリファレンスの[DescribeAutoScalingInstances](#)「」の「」を参照してください。

## DescribeScalingActivities

次のコード例は、DescribeScalingActivities を使用する方法を示しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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::
```



```

    // 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::

```

- API 詳細については、AWS SDK Rust API リファレンスの [DescribeScalingActivities](#) 「」の「」を参照してください。

## DisableMetricsCollection

次のコード例は、DisableMetricsCollection を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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 詳細については、AWS SDK Rust APIリファレンスの[DisableMetricsCollection](#)「」の「」を参照してください。

## EnableMetricsCollection

次のコード例は、EnableMetricsCollection を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [EnableMetricsCollection](#) 「」の「」を参照してください。

## SetDesiredCapacity

次の例は、SetDesiredCapacity を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [SetDesiredCapacity](#) 「」の「」を参照してください。

## TerminateInstanceInAutoScalingGroup

次の例は、`TerminateInstanceInAutoScalingGroup` を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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"))
    }
}
```

```
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 詳細については、AWS SDK Rust API リファレンスの [TerminateInstanceInAutoScalingGroup](#) 「」の「」を参照してください。

## UpdateAutoScalingGroup

次のコード例は、UpdateAutoScalingGroup を使用する方法を示しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [UpdateAutoScalingGroup](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon Bedrock ランタイムの例

次のコード例は、Amazon Bedrock ランタイムで for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### トピック

- [Anthropic Claude](#)

# Anthropic Claude

## Converse

次のコード例は、Bedrock の Converse を使用して Anthropic Claude にテキストメッセージを送信する方法を示していますAPI。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

Bedrock の Converse を使用して、Anthropic Claude にテキストメッセージを送信しますAPI。

```
#[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)
}

```

ステートメント、エラーユーティリティ、定数を使用します。

```

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 詳細については、AWS SDK 「Rust API [リファレンスの](#)」の「[Converse in](#)」を参照してください。

## ConverseStream

次のコード例は、Bedrock の Converse を使用して Anthropic Claude にテキストメッセージを送信 API し、レスポンスストリームをリアルタイムで処理する方法を示しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

Bedrock のを使用して、Anthropic Claude にテキストメッセージを送信し、返信トークンをストリーミングします ConverseStream API。

```
#[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(),
    })
}

```

ステートメント、エラーユーティリティ、定数を使用します。

```

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 詳細については、AWS SDK Rust API リファレンスの [ConverseStream](#) 「」の「」を参照してください。

### シナリオ: Converse でのツールの使用 API

次のコード例は、アプリケーション、生成 AI モデル、接続されたツール間の一般的なインタラクションを構築する方法、または AI と外部環境間のインタラクションを仲介 APIs する方法を示しています。外部気象を API AI モデルに接続する例を使用して、ユーザー入力に基づいてリアルタイムの気象情報を提供できるようにします。

### SDK Rust の

#### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

デモのプライマリシナリオとロジック。これにより、ユーザー、Amazon Bedrock Converse API、および気象ツール間の会話がオーケストレーションされます。

```

#[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();
}
}

```

デモで使用される気象ツール。このスクリプトはツール仕様を定義し、Open-Meteo から を使用して気象データを取得するロジックを実装しますAPI。

```

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()?)
}

```

メッセージコンテンツブロックを出力するユーティリティ。

```

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:?})"
        )))
    }
}
}

```

ステートメント、エラーユーティリティ、定数を使用します。

```

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 詳細については、AWS SDK 「Rust API [リファレンスの](#)」の「[Converse in](#)」を参照してください。

## SDK for Rust を使用した Amazon Cognito ID プロバイダーの例

次のコード例は、Amazon Cognito ID プロバイダーで for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

### アクション

#### ListUserPools

次の例は、ListUserPools を使用する方法を説明しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListUserPools](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon Cognito Sync の例

次のコード例は、Amazon Cognito Sync で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### ListIdentityPoolUsage

次のコード例は、ListIdentityPoolUsage を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListIdentityPoolUsage](#) 「」の「」を参照してください。

## SDK for Rust を使用した Firehose の例

次のコード例は、for Rust with Firehose を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### トピック

- [アクション](#)

## アクション

### PutRecordBatch

次のコード例は、PutRecordBatch を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [PutRecordBatch](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon DocumentDB の例

次のコード例は、Amazon DocumentDB で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [サーバーレスサンプル](#)

## サーバーレスサンプル

Amazon DocumentDB トリガーから Lambda 関数を呼び出す

次のコード例は、DocumentDB 変更ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DocumentDB ペイロードを取得し、レコードの内容をログ記録します。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で Amazon DocumentDB イベントの消費。

```
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 Rust を使用した DynamoDB の例

次のコード例は、DynamoDB で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1つのサービス内から、または他のAWSのサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS コミュニティへの貢献は、複数のチームが作成し、維持している例です AWS。フィードバックを提供するには、リンクされたリポジトリで提供されているメカニズムを使用します。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)
- [シナリオ](#)
- [サーバーレスサンプル](#)
- [AWS コミュニティへの貢献](#)

## アクション

### CreateTable

次の例は、CreateTable を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateTable](#) 「」の「」を参照してください。

## DeleteItem

次のコード例は、DeleteItem を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteItem](#) 「」の「」を参照してください。

## DeleteTable

次の例は、DeleteTable を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteTable](#) 「」の「」を参照してください。

## ListTables

次のコード例は、ListTables を使用する方法を示しています。



## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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::
```

テーブルが存在するかどうかを確認します。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListTables](#) 「」の「」を参照してください。

## PutItem

次の例は、PutItem を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 {username}, {first_name} {last_name}, age {age} as {p_type} user",
```

```

        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}

```

- API 詳細については、AWS SDK Rust API リファレンスの [PutItem](#) 「」の「」を参照してください。

## Query

次の例は、Query を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

指定した年に作られた映画を検索します。

```

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 詳細については、AWS SDK「Rust APIリファレンスの [クエリ](#)」を参照してください。

## Scan

次の例は、Scan を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK「Rust APIリファレンスの [スキャン](#)」を参照してください。

## シナリオ

### ローカルインスタンスに接続する

次のコード例は、エンドポイントを上書きURLして DynamoDB と のローカル開発デプロイに接続する方法を示しています AWS SDK。

詳細については、「[DynamoDB Local](#)」を参照してください。

### SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// 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:?}"),
}
}
```

## サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

### SDK Rust 用の

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## PartiQL を使用してテーブルに対してクエリを実行する

次のコードサンプルは、以下の操作方法を示しています。

- SELECT ステートメントを実行して項目を取得します。
- INSERT ステートメントを実行して項目を追加します。
- UPDATE ステートメントを実行して項目を更新します。
- DELETE ステートメントを実行して項目を削除します。

### SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 "{}" WHERE "{}" = ?"#))
        .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 詳細については、AWS SDK Rust API リファレンスの [ExecuteStatement](#) 「」の「」を参照してください。

## EXIF およびその他のイメージ情報を保存する

次のコードサンプルは、以下の操作方法を示しています。

- JPG、JPEG、または PNG ファイルから EXIF 情報を取得します。
- Amazon S3 バケットにイメージファイルをアップロードします。
- Amazon Rekognition を使用して、ファイル内の 3 つの上位属性 (ラベル) を特定します。
- リージョンの Amazon DynamoDB テーブルに EXIF および ラベル情報を追加します。

### SDK Rust 用の

JPG、JPEG または PNG ファイルから EXIF 情報を取得し、イメージファイルを Amazon S3 バケットにアップロードし、Amazon Rekognition を使用してファイル内の 3 つの上位属性 (Amazon Rekognition のラベル) を識別し、EXIF および ラベル情報をリージョンの Amazon DynamoDB テーブルに追加します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3

## サーバーレスサンプル

### DynamoDB トリガーから Lambda 関数を呼び出す

次のコード例は、DynamoDB ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DynamoDB ペイロードを取得し、レコードの内容をログ記録します。

## SDK Rust 用の

 Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプルリポジトリ](#)で完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で DynamoDB イベントを利用します。

```
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 トリガーで Lambda 関数のバッチアイテムの失敗をレポートする

次のコード例は、DynamoDB ストリームからイベントを受信する Lambda 関数に部分的なバッチレスポンスを実装する方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

### SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
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 コミュニティへの貢献

### サーバーレスアプリケーションの構築とテスト

次のコード例は、Lambda と DynamoDB で API Gateway を使用してサーバーレスアプリケーションを構築およびテストする方法を示しています。

### SDK Rust 用の

Rust を使用して、Lambda と DynamoDB を備えた API ゲートウェイで構成されるサーバーレスアプリケーションを構築してテストする方法を示します SDK。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda

## SDK for Rust を使用した Amazon EBS の例

次のコード例は、Amazon で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています EBS。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### CompleteSnapshot

次の例は、CompleteSnapshot を使用する方法を説明しています。

SDK Rust 用の

#### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CompleteSnapshot](#) 「」の「」を参照してください。

## PutSnapshotBlock

次のコード例は、PutSnapshotBlock を使用する方法を示しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [PutSnapshotBlock](#) 「」の「」を参照してください。

## StartSnapshot

次のコード例は、StartSnapshot を使用する方法を示しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust APIリファレンスの[StartSnapshot](#)「」の「」を参照してください。

## SDK for Rust を使用した Amazon EC2の例

次のコード例は、Amazon で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示していますEC2。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### 開始方法

#### Amazon へようこそ EC2

次のコード例は、Amazon の使用を開始する方法を示していますEC2。

#### SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeSecurityGroups](#) 「」の「」を参照してください。

## トピック

- [基本](#)
- [アクション](#)

## 基本

### 基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- キーペアとセキュリティグループを作成します。
- Amazon マシンイメージ (AMI) と互換性のあるインスタンスタイプを選択し、インスタンスを作成します。
- インスタンスを停止し、再起動します。
- Elastic IP アドレスをインスタンスに関連付ける。

- を使用してインスタンスに接続しSSH、リソースをクリーンアップします。

## SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

EC2InstanceScenario 実装には、例全体を実行するロジックが含まれています。

```
#!/ 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 構造体はテスト用のオートモックポイントとして機能し、その関数はEC2SDK呼び出しをラップします。

```

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 構造体はテスト用のオートモックポイントとして機能し、その関数はSSMSDK呼び出しをラップします。

```

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:::<Result<Vec<Parameter>, _>>()?;
        Ok(params)
    }
}

```

このシナリオでは、複数の「Manager」スタイルの構造体を使用して、シナリオ全体で作成および削除されたリソースへのアクセスを処理します。

```

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."),
}
}
```

シナリオの主なエントリポイント。

```
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 詳細については、AWS SDK Rust API リファレンスの [以下のトピックを参照してください](#)。
  - [AllocateAddress](#)
  - [AssociateAddress](#)
  - [AuthorizeSecurityGroupIngress](#)
  - [CreateKeyPair](#)
  - [CreateSecurityGroup](#)
  - [DeleteKeyPair](#)

- [DeleteSecurityGroup](#)
- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

## アクション

### AllocateAddress

次の例は、AllocateAddress を使用する方法を説明しています。

SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
pub async fn allocate_ip_address(&self) -> Result<AllocateAddressOutput,
EC2Error> {
    self.client
        .allocate_address()
        .domain(DomainType::Vpc)
        .send()
        .await
        .map_err(EC2Error::from)
```

```
}
```

- API 詳細については、[AllocateAddress](#) 「」のAWS SDK「Rust APIリファレンス」を参照してください。

## AssociateAddress

次のコード例は、AssociateAddress を使用する方法を示しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust APIリファレンスの[AssociateAddress](#) 「」の「」を参照してください。

## AuthorizeSecurityGroupIngress

次の例は、AuthorizeSecurityGroupIngress を使用する方法を説明しています。

## SDK Rust 用の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// 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 詳細については、AWS SDK Rust API リファレンスの [AuthorizeSecurityGroupIngress](#) 「」の「」を参照してください。

## CreateKeyPair

次の例は、CreateKeyPair を使用する方法を説明しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

EC2 クライアントの create\_key\_pair を呼び出し、返されたマテリアルを抽出する Rust 実装。

```
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\_impl を呼び出し、PEMプライベートキーを安全に保存する関数。

```
/// 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 詳細については、AWS SDK Rust API リファレンスの [CreateKeyPair](#) 「」の「」を参照してください。

## CreateSecurityGroup

次の例は、CreateSecurityGroup を使用する方法を説明しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateSecurityGroup](#) 「」の「」を参照してください。

## CreateTags

次のコード例は、CreateTags を使用する方法を示しています。

## SDK Rust 用の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

この例では、インスタンスの作成後に Name タグを適用します。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateTags](#) 「」の「」を参照してください。

## DeleteKeyPair

次のコード例は、DeleteKeyPair を使用する方法を示しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

バックアッププライベートキーも削除する `delete_PEMkey` のラッパー。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteKeyPair](#) 「」の「」を参照してください。

## DeleteSecurityGroup

次のコード例は、DeleteSecurityGroup を使用する方法を示しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteSecurityGroup](#) 「」の「」を参照してください。

## DeleteSnapshot

次のコード例は、DeleteSnapshot を使用する方法を示しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
async fn delete_snapshot(client: &Client, id: &str) -> Result<(), Error> {
    client.delete_snapshot().snapshot_id(id).send().await?;

    println!("Deleted");

    Ok(())
}
```

- API 詳細については、AWS SDK Rust API リファレンスの [DeleteSnapshot](#) 「」の「」を参照してください。

## DescribeImages

次のコード例は、DescribeImages を使用する方法を示しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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)
    }
}
```

で list\_images 関数を使用してSSM、環境に基づいて を制限します。の詳細についてはSSM、 [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) を参照してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeImages](#) 「」の「」を参照してください。

## DescribeInstanceStatus

次の例は、DescribeInstanceStatus を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeInstanceStatus](#) 「」の「」を参照してください。

## DescribeInstanceTypes

次のコード例は、DescribeInstanceTypes を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// 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 詳細については、AWS SDK Rust API リファレンスの [DescribeInstanceTypes](#) 「」の「」を参照してください。

## DescribeInstances

次の例は、DescribeInstances を使用する方法を説明しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

EC2 インスタンスの詳細を取得します。

```
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 インスタンスを作成したら、その詳細を取得して保存します。

```
/// 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 詳細については、AWS SDK Rust API リファレンスの [DescribeInstances](#) 「」の「」を参照してください。

## DescribeKeyPairs

次の例は、DescribeKeyPairs を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeKeyPairs](#) 「」の「」を参照してください。

## DescribeRegions

次の例は、DescribeRegions を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの[DescribeRegions](#)「」の「」を参照してください。

## DescribeSecurityGroups

次の例は、DescribeSecurityGroups を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeSecurityGroups](#) 「」の「」を参照してください。

## DescribeSnapshots

次の例は、DescribeSnapshots を使用する方法を説明しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

スナップショットの状態を表示します。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeSnapshots](#) 「」の「」を参照してください。

## DisassociateAddress

次の例は、DisassociateAddress を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
pub async fn disassociate_ip_address(&self, association_id: &str) -> Result<(),
EC2Error> {
    self.client
        .disassociate_address()
        .association_id(association_id)
        .send()
        .await?;
    Ok(())
}
```

- API 詳細については、AWS SDK Rust API リファレンスの [DisassociateAddress](#) 「」の「」を参照してください。

## RebootInstances

次のコード例は、RebootInstances を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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(())
}
```

Waiters を使用して、インスタンスが停止状態および準備完了状態になるウェーターAPI。ウェーターAPIを使用するには、rust ファイルで「aws\_sdk\_ec2::client::Waiters」を使用する必要があります。



```
/// 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 詳細については、AWS SDK Rust API リファレンスの [RebootInstances](#) 「」の「」を参照してください。

## ReleaseAddress

次の例は、ReleaseAddress を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
pub async fn deallocate_ip_address(&self, allocation_id: &str) -> Result<(),
EC2Error> {
    self.client
        .release_address()
        .allocation_id(allocation_id)
        .send()
        .await?;
    Ok(())
}
```

- API 詳細については、AWS SDK Rust API リファレンスの [ReleaseAddress](#) 「」の「」を参照してください。

## RunInstances

次の例は、RunInstances を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [RunInstances](#) 「」の「」を参照してください。

## StartInstances

次のコード例は、StartInstances を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

インスタンス ID EC2 でインスタンスを起動します。

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

Waiters を使用して、インスタンスが準備完了状態およびステータス ok 状態になるまで待ちます API。ウェーターAPIを使用するには、rust ファイルで「aws\_sdk\_ec2::client::Waiters」を使用する必要があります。

```
/// 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 詳細については、AWS SDK Rust APIリファレンスの[StartInstances](#)「」の「」を参照してください。

## StopInstances

次の例は、StopInstances を使用する方法を説明しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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(())
}
```

Waiters を使用して、インスタンスが停止状態になるまで待ちますAPI。ウェーターAPIを使用するには、rust ファイルで「aws\_sdk\_ec2::client::Waiters」を使用する必要があります。

```
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 詳細については、AWS SDK Rust API リファレンスの [StopInstances](#) 「」の「」を参照してください。

## TerminateInstances

次の例は、`TerminateInstances` を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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(())
}
```

`Waiters` を使用して、インスタンスが終了状態になるまで待ちますAPI。ウェーターAPIを使用するには、rust ファイルで「`aws_sdk_ec2::client::Waiters`」を使用する必要があります。

```
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 詳細については、AWS SDK Rust API リファレンスの [TerminateInstances](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon ECR の例

次のコード例は、Amazon で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています ECR。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### DescribeRepositories

次のコード例は、DescribeRepositories を使用する方法を示しています。



## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeRepositories](#) 「」の「」を参照してください。

## ListImages

次のコード例は、ListImages を使用する方法を示しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListImages](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon ECS の例

次のコード例は、Amazon で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています ECS。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

## トピック

- [アクション](#)

## アクション

### CreateCluster

次の例は、CreateCluster を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateCluster](#) 「」の「」を参照してください。

### DeleteCluster

次のコード例は、DeleteCluster を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteCluster](#) 「」の「」を参照してください。

## DescribeClusters

次のコード例は、DescribeClusters を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeClusters](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon EKS の例

次のコード例は、Amazon で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示していますEKS。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### CreateCluster

次のコード例は、CreateCluster を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateCluster](#) 「」の「」を参照してください。

## DeleteCluster

次のコード例は、DeleteCluster を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteCluster](#) 「」の「」を参照してください。

## AWS Glue SDK for Rust を使用した の例

次のコード例は、`for Rust` を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています AWS Glue。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### 開始方法

こんにちは AWS Glue は

次のコード例は、AWS Glue の使用を開始する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListJobs](#) 「」の「」を参照してください。

## トピック

- [基本](#)
- [アクション](#)

## 基本

### 基本を学ぶ


次のコードサンプルは、以下の操作方法を示しています。

- パブリック Amazon S3 バケットをクローलし、CSV形式のメタデータのデータベースを生成するクローラーを作成します。
- のデータベースとテーブルに関する情報を一覧表示します AWS Glue Data Catalog。
- S3 バケットから CSV データを抽出し、データを変換して、JSON形式の出力を別の S3 バケットにロードするジョブを作成します。
- ジョブ実行に関する情報を一覧表示し、変換されたデータを表示してリソースをクリーンアップする。

詳細については、[「チュートリアル: AWS Glue Studio の開始方法」](#)を参照してください。



## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

パブリック Amazon Simple Storage Service (Amazon S3) バケットをクローलし、見つかった CSV形式のデータを記述するメタデータデータベースを生成するクローラを作成して実行します。

```
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)),
    }
}
}
}??;
```

のデータベースとテーブルに関する情報を一覧表示します 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();
```

ソース Amazon S3 バケットからCSVデータを抽出し、フィールドを削除および名前を変更して変換し、JSON形式の出力を別の Amazon S3 バケットにロードするジョブを作成して実行します。

```
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();
```

デモによって作成されたすべてのリソースを削除します。

```
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 詳細については、AWS SDK Rust API リファレンスの以下のトピックを参照してください。
  - [CreateCrawler](#)
  - [CreateJob](#)
  - [DeleteCrawler](#)
  - [DeleteDatabase](#)
  - [DeleteJob](#)
  - [DeleteTable](#)
  - [GetCrawler](#)
  - [GetDatabase](#)
  - [GetDatabases](#)
  - [GetJob](#)

- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

## アクション

### CreateCrawler

次の例は、CreateCrawler を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateCrawler](#) 「」の「」を参照してください。

## CreateJob

次のコード例は、CreateJob を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateJob](#) 「」の「」を参照してください。

## DeleteCrawler

次のコード例は、DeleteCrawler を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
glue.delete_crawler()
    .name(self.crawler())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- API 詳細については、AWS SDK Rust API リファレンスの [DeleteCrawler](#) 「」の「」を参照してください。

## DeleteDatabase

次の例は、DeleteDatabase を使用する方法を説明しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
glue.delete_database()
    .name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- API 詳細については、AWS SDK Rust API リファレンスの [DeleteDatabase](#) 「」の「」を参照してください。

## DeleteJob

次の例は、DeleteJob を使用する方法を説明しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
glue.delete_job()
    .job_name(self.job())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- API 詳細については、AWS SDK Rust API リファレンスの [DeleteJob](#) 「」の「」を参照してください。



## DeleteTable

次の例は、DeleteTable を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
for t in &self.tables {
    glue.delete_table()
        .name(t.name())
        .database_name(self.database())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;
}
```

- API 詳細については、AWS SDK Rust API リファレンスの[DeleteTable](#)「」の「」を参照してください。

## GetCrawler

次の例は、GetCrawler を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
let tmp_crawler = glue
    .get_crawler()
    .name(self.crawler())
```

```
.send()  
.await  
.map_err(GlueMvpError::from_glue_sdk)?;
```

- API 詳細については、AWS SDK Rust API リファレンスの [GetCrawler](#) 「」の「」を参照してください。

## GetDatabase

次のコード例は、GetDatabase を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GetDatabase](#) 「」の「」を参照してください。

## GetJobRun

次の例は、GetJobRun を使用する方法を説明しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの[GetJobRun](#)「」の「」を参照してください。

## GetTables

次のコード例は、GetTables を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
let tables = glue
    .get_tables()
    .database_name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

let tables = tables.table_list();
```

- API 詳細については、AWS SDK Rust API リファレンスの[GetTables](#)「」の「」を参照してください。

## ListJobs

次のコード例は、ListJobs を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListJobs](#) 「」の「」を参照してください。

## StartCrawler

次のコード例は、StartCrawler を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [StartCrawler](#) 「」の「」を参照してください。

## StartJobRun

次のコード例は、StartJobRun を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [StartJobRun](#) 「」の「」を参照してください。

## IAM SDK for Rust を使用した の例

次のコード例は、で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示していますIAM。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### 開始方法

Hello IAM

次のコード例は、IAM の使用を開始する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

src/bin/hello.rs から。

```
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.rs から。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListPolicies](#) 「」の「」を参照してください。

## トピック

- [基本](#)
- [アクション](#)

## 基本

### 基本を学ぶ

次のコードサンプルは、ユーザーを作成してロールを割り当てる方法を示しています。

#### Warning

セキュリティリスクを回避するため、専用ソフトウェアの開発時や実際のデータの使用時に、認証に IAM ユーザーを使用しないでください。代わりに、[AWS IAM Identity Center](#) などの ID プロバイダーとのフェデレーションを使用してください。

- 権限のないユーザーを作成します。
- 指定したアカウントに Amazon S3 バケットへのアクセス権限を付与するロールを作成します。
- ユーザーにロールを引き受けさせるポリシーを追加します。
- ロールを引き受け、一時的な認証情報を使用して S3 バケットを一覧表示しリソースをクリーンアップします。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの以下のトピックを参照してください。
  - [AttachRolePolicy](#)
  - [CreateAccessKey](#)
  - [CreatePolicy](#)
  - [CreateRole](#)
  - [CreateUser](#)
  - [DeleteAccessKey](#)
  - [DeletePolicy](#)
  - [DeleteRole](#)
  - [DeleteUser](#)
  - [DeleteUserPolicy](#)
  - [DetachRolePolicy](#)
  - [PutUserPolicy](#)

## アクション

### AttachRolePolicy

次の例は、AttachRolePolicy を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「 」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [AttachRolePolicy](#) 「」の「」を参照してください。

## AttachUserPolicy

次の例は、AttachUserPolicy を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [AttachUserPolicy](#) 「」の「」を参照してください。



## CreateAccessKey

次の例は、CreateAccessKey を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateAccessKey](#) 「」の「」を参照してください。

## CreatePolicy

次のコード例は、CreatePolicy を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの[CreatePolicy](#)「」の「」を参照してください。

## CreateRole

次のコード例は、CreateRole を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateRole](#) 「」の「」を参照してください。

## CreateServiceLinkedRole

次のコード例は、CreateServiceLinkedRole を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateServiceLinkedRole](#) 「」の「」を参照してください。

## CreateUser

次のコード例は、CreateUser を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateUser](#) 「」の「」を参照してください。

## DeleteAccessKey

次の例は、DeleteAccessKey を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteAccessKey](#) 「」の「」を参照してください。

## DeletePolicy

次の例は、DeletePolicy を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
pub async fn delete_policy(client: &iamClient, policy: Policy) -> Result<(),
iamError> {
    client
        .delete_policy()
        .policy_arn(policy.arn.unwrap())
        .send()
        .await?;
    Ok(())
}
```

- API 詳細については、AWS SDK Rust API リファレンスの[DeletePolicy](#)「」の「」を参照してください。

## DeleteRole

次の例は、DeleteRole を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteRole](#) 「」の「」を参照してください。

## DeleteServiceLinkedRole

次の例は、DeleteServiceLinkedRole を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust APIリファレンスの[DeleteServiceLinkedRole](#)「」の「」を参照してください。

## DeleteUser

次のコード例は、DeleteUser を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteUser](#) 「」の「」を参照してください。

## DeleteUserPolicy

次のコード例は、DeleteUserPolicy を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteUserPolicy](#) 「」の「」を参照してください。

## DetachRolePolicy

次のコード例は、DetachRolePolicy を使用する方法を示しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DetachRolePolicy](#) 「」の「」を参照してください。

**DetachUserPolicy**

次の例は、DetachUserPolicy を使用する方法を説明しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DetachUserPolicy](#) 「」の「」を参照してください。

## GetAccountPasswordPolicy

次の例は、GetAccountPasswordPolicy を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GetAccountPasswordPolicy](#) 「」の「」を参照してください。

## GetRole

次の例は、GetRole を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GetRole](#) 「」の「」を参照してください。

## ListAttachedRolePolicies

次の例は、ListAttachedRolePolicies を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListAttachedRolePolicies](#) 「」の「」を参照してください。

## ListGroups

次のコード例は、ListGroups を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
pub async fn list_groups(
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
```

```
) -> Result<ListGroupOutput, SdkError<ListGroupError>> {
    let response = client
        .list_groups()
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- API 詳細については、AWS SDK Rust API リファレンスの [ListGroup](#) 「」の「」を参照してください。

## ListPolicies

次のコード例は、ListPolicies を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListPolicies](#) 「」の「」を参照してください。

## ListRolePolicies

次のコード例は、ListRolePolicies を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListRolePolicies](#) 「」の「」を参照してください。

## ListRoles

次の例は、ListRoles を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListRoles](#) 「」の「」を参照してください。

## ListSAMLProviders

次のコード例は、ListSAMLProviders を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
pub async fn list_saml_providers(
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}
```

- API 詳細については、Rust リファレンスの「」の [ListSAMLProviders](#)」を参照してください。  
AWS SDK API

## ListUsers

次の例は、ListUsers を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListUsers](#) 「」の「」を参照してください。

## AWS IoT SDK for Rust を使用した の例

次のコード例は、`for Rust` を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています AWS IoT。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### DescribeEndpoint

次のコード例は、`DescribeEndpoint` を使用する方法を示しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeEndpoint](#) 「」の「」を参照してください。

**ListThings**

次のコード例は、ListThings を使用する方法を示しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListThings](#) 「」の「」を参照してください。

## SDK for Rust を使用した Kinesis の例

次のコード例は、Kinesis で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### トピック

- [アクション](#)
- [サーバーレスサンプル](#)

## アクション

### CreateStream

次のコード例は、CreateStream を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateStream](#) 「」の「」を参照してください。

### DeleteStream

次の例は、DeleteStream を使用する方法を説明しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- API 詳細については、[DeleteStream](#) 「」のAWS SDK 「Rust APIリファレンス」を参照してください。

## DescribeStream

次の例は、DescribeStream を使用する方法を説明しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeStream](#) 「」の「」を参照してください。

## ListStreams

次のコード例は、ListStreams を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListStreams](#) 「」の「」を参照してください。

## PutRecord

次の例は、PutRecord を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [PutRecord](#) 「」の「」を参照してください。

## サーバーレスサンプル

Kinesis トリガーから Lambda 関数を呼び出す

次のコード例では、Kinesis ストリームからレコードを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数は Kinesis ペイロードを取得し、それを Base64 からデコードして、そのレコードの内容をログ記録します。



## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用した Lambda での Kinesis イベントの消費。

```
// 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 トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、Kinesis ストリームからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

### Rust を使用した Lambda での Kinesis バッチアイテム失敗のレポート。

```

// 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 Rust を使用した の例

次のコード例は、`aws-kms` で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています AWS KMS。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### トピック

- [アクション](#)

## アクション

### CreateKey

次のコード例は、CreateKey を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、[CreateKey](#)「」のAWS SDK「Rust APIリファレンス」を参照してください。

### Decrypt

次のコード例は、Decrypt を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK「Rust APIリファレンスの [復号化](#)」を参照してください。

## Encrypt

次の例は、Encrypt を使用する方法を説明しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK 「Rust APIリファレンスの [暗号化](#)」を参照してください。

## GenerateDataKey

次のコード例は、GenerateDataKey を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GenerateDataKey](#) 「」の「」を参照してください。

## GenerateDataKeyWithoutPlaintext

次の例は、GenerateDataKeyWithoutPlaintext を使用する方法を説明しています。



## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GenerateDataKeyWithoutPlaintext](#) 「」の「」を参照してください。

## GenerateRandom

次のコード例は、GenerateRandom を使用する方法を示しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GenerateRandom](#) 「」の「」を参照してください。

**ListKeys**

次のコード例は、ListKeys を使用する方法を示しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListKeys](#) 「」の「」を参照してください。

**ReEncrypt**

次のコード例は、ReEncrypt を使用する方法を示しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ReEncrypt](#) 「」の「」を参照してください。

## SDK for Rust を使用した Lambda の例

次のコード例は、Lambda で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS コミュニティへの貢献は、複数のチームが作成し、維持している例です AWS。フィードバックを提供するには、リンクされたリポジトリで提供されているメカニズムを使用します。

各例には、完全なソースコードへのリンクが含まれており、そこからコードの設定方法と実行方法に関する手順を確認できます。

### トピック

- [基本](#)
- [アクション](#)
- [シナリオ](#)
- [サーバーレスサンプル](#)
- [AWS コミュニティへの貢献](#)

## 基本

### 基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- IAM ロールと Lambda 関数を作成し、ハンドラーコードをアップロードします。
- 1つのパラメーターで関数を呼び出して、結果を取得します。

- 関数コードを更新し、環境変数で設定します。
- 新しいパラメーターで関数を呼び出して、結果を取得します。返された実行ログを表示します。
- アカウントの関数を一覧表示し、リソースをクリーンアップします。

詳細については、「[コンソールで Lambda 関数を作成する](#)」を参照してください。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

このシナリオで使用した依存関係を含む Cargo.toml。

```
[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"
```

このシナリオの Lambda 呼び出しを効率化するユーティリティのコレクション。このファイルはクレート内の `src/ations.rs` というファイルです。

```
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::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)
}

/**
 * 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(),
        );
    }
}

```

コマンドラインフラグを使用して一部の動作を制御し、シナリオを最初から最後まで実行するためのバイナリ。このファイルはクレート内の `src/bin/scenario.rs` です。

```

/*
## 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 詳細については、AWS SDK Rust APIリファレンスの [以下のトピックを参照してください](#)。
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## アクション

### CreateFunction

次の例は、CreateFunction を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/**
 * 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 詳細については、AWS SDK Rust API リファレンスの [CreateFunction](#) 「」の「」を参照してください。

## DeleteFunction

次のコード例は、DeleteFunction を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/** 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 詳細については、AWS SDK Rust APIリファレンスの[DeleteFunction](#)「」の「」を参照してください。

## GetFunction

次のコード例は、GetFunction を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/** 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 詳細については、AWS SDK Rust API リファレンスの[GetFunction](#)「」の「」を参照してください。

## Invoke

次の例は、Invoke を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/** 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 詳細については、AWS SDK 「Rust API [リファレンスのために で呼び出す](#)」を参照してください。

## ListFunctions

次の例は、ListFunctions を使用する方法を説明しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/** 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 詳細については、AWS SDK Rust API リファレンスの [ListFunctions](#) 「」の「」を参照してください。

**UpdateFunctionCode**

次のコード例は、UpdateFunctionCode を使用する方法を示しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/** 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 詳細については、AWS SDK Rust API リファレンスの [UpdateFunctionCode](#) 「」の「」を参照してください。

## UpdateFunctionConfiguration

次のコード例は、UpdateFunctionConfiguration を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/** 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 詳細については、AWS SDK Rust API リファレンスの [UpdateFunctionConfiguration](#) 「」の「」を参照してください。

## シナリオ

### サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

#### SDK Rust の

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例のソースについて詳しくは、[AWS コミュニティ](#) でブログ投稿を参照してください。

#### この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## サーバーレスサンプル

### Lambda 関数での Amazon RDS データベースへの接続

次のコード例は、RDS データベースに接続する Lambda 関数を実装する方法を示しています。この関数は、シンプルなデータベースリクエストを実行し、結果を返します。

#### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda 関数の Amazon RDS データベースに接続する。

```
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 トリガーから Lambda 関数を呼び出す

次のコード例では、Kinesis ストリームからレコードを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数は Kinesis ペイロードを取得し、それを Base64 からデコードして、そのレコードの内容をログ記録します。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用した Lambda での Kinesis イベントの消費。

```
// 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 トリガーから Lambda 関数を呼び出す

次のコード例は、DynamoDB ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DynamoDB ペイロードを取得し、レコードの内容をログ記録します。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で DynamoDB イベントを利用します。

```
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 トリガーから Lambda 関数を呼び出す

次のコード例は、DocumentDB 変更ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DocumentDB ペイロードを取得し、レコードの内容をログ記録します。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で Amazon DocumentDB イベントの消費。

```
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 S3 トリガーから Lambda 関数を呼び出す

次のコード例は、S3 バケットにオブジェクトをアップロードすることによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。関数は、イベントパラメータから S3 バケット名とオブジェクトキーを取得し、Amazon S3 API を呼び出してオブジェクトのコンテンツタイプを取得してログに記録します。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で S3 イベントを消費します。

```
// 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トリガーから Lambda 関数を呼び出す

次のコード例は、SNSトピックからメッセージを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用した Lambda での SNS イベントの消費。

```
// 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トリガーから Lambda 関数を呼び出す

次のコード例は、SQSキューからメッセージを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用した Lambda でのSQSイベントの消費。

```
// 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 トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、Kinesis ストリームからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

## Rust を使用した Lambda での Kinesis バッチアイテム失敗のレポート。

```

// 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 トリガーで Lambda 関数のバッチアイテムの失敗をレポートする

次のコード例は、DynamoDB ストリームからイベントを受信する Lambda 関数に部分的なバッチレスポンスを実装する方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

## Rust を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
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
}
```

## Amazon SQSトリガーを使用した Lambda 関数のバッチアイテム失敗のレポート

次のコード例は、SQSキューからイベントを受信する Lambda 関数に部分的なバッチレスポンスを実装する方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda でSQSバッチアイテムの失敗を報告する。

```
// 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 コミュニティへの貢献

### サーバーレスアプリケーションの構築とテスト

次のコード例は、Lambda と DynamoDB で API Gateway を使用してサーバーレスアプリケーションを構築およびテストする方法を示しています。

#### SDK Rust の

Rust を使用して、Lambda と DynamoDB を備えた API ゲートウェイで構成されるサーバーレスアプリケーションを構築してテストする方法を示します SDK。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda

## MediaLive SDK for Rust を使用した の例

次のコード例は、で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています MediaLive。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

#### トピック

- [アクション](#)

## アクション

### ListInputs

次のコード例は、ListInputs を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

リージョン MediaLive の入力名と ARNs を一覧表示します。

```
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 詳細については、AWS SDK Rust API リファレンスの[ListInputs](#)「」の「」を参照してください。

## MediaPackage SDK for Rust を使用した の例

次のコード例は、`for Rust` を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています MediaPackage。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### ListChannels

次の例は、ListChannels を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

チャンネルARNsと説明を一覧表示します。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListChannels](#) 「」の「」を参照してください。

## ListOriginEndpoints

次の例は、ListOriginEndpoints を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

エンドポイントの説明とを一覧表示しますURLs。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListOriginEndpoints](#) 「」の「」を参照してください。

# SDK for Rust を使用した Amazon Polly の例

次のコード例は、Amazon Polly で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1つのサービス内から、または他のAWSのサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)
- [シナリオ](#)

## アクション

### DescribeVoices

次の例は、DescribeVoices を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeVoices](#) 「」の「」を参照してください。

## ListLexicons

次の例は、ListLexicons を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListLexicons](#) 「」の「」を参照してください。

## PutLexicon

次の例は、PutLexicon を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

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 詳細については、AWS SDK Rust API リファレンスの [PutLexicon](#) 「」の「」を参照してください。

## SynthesizeSpeech

次の例は、SynthesizeSpeech を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [SynthesizeSpeech](#) 「」の「」を参照してください。

## シナリオ

テキストを音声に変換し、テキストに戻す

次のコードサンプルは、以下の操作方法を示しています。

- Amazon Polly を使用して、プレーンテキスト (UTF-8) 入力ファイルをオーディオファイルに合成します。
- Amazon S3 バケットに音声ファイルをアップロードします。
- Amazon Transcribe を使用して、音声ファイルをテキストに変換します。
- テキストを表示します。

## SDK Rust の

Amazon Polly を使用して、プレーンテキスト (UTF-8) 入カファイルをオーディオファイルに合成し、オーディオファイルを Amazon S3 バケットにアップロードし、Amazon Transcribe を使用してそのオーディオファイルをテキストに変換し、テキストを表示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- Amazon Polly
- Amazon S3
- Amazon Transcribe

## QLDB SDK for Rust を使用した の例

次のコード例は、`for Rust` を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示していますQLDB。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### CreateLedger

次の例は、CreateLedger を使用する方法を説明しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateLedger](#) 「」の「」を参照してください。

## ListLedgers

次のコード例は、ListLedgers を使用する方法を示しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListLedgers](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon RDS の例

次のコード例は、Amazon で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています RDS。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [サーバーレスサンプル](#)

### サーバーレスサンプル

Lambda 関数での Amazon RDS データベースへの接続

次のコード例は、RDS データベースに接続する Lambda 関数を実装する方法を示しています。この関数は、シンプルなデータベースリクエストを実行し、結果を返します。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

## Rust を使用して Lambda 関数の Amazon RDS データベースに接続する。

```
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 Rust を使用した Amazon RDS Data Service の例

次のコード例は、Amazon RDS Data Service で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### ExecuteStatement

次の例は、ExecuteStatement を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ExecuteStatement](#) 「」の「」を参照してください。

# SDK for Rust を使用した Amazon Rekognition の例

次のコード例は、Amazon Rekognition で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1つのサービス内から、または他のAWSのサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [シナリオ](#)

## シナリオ

サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

SDK Rust の

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## イメージ内の顔を検出します

次のコードサンプルは、以下の操作方法を示しています。

- イメージを Amazon S3 バケットに保存します。
- Amazon Rekognition を使用して、年齢層、性別、感情 (笑顔など) などの顔の詳細を検出します。
- これらの詳細を表示します。

### SDK Rust の

アップロードプレフィックスを付け、Amazon S3 バケット内でイメージを保存し、Amazon Rekognition を使用して、年齢層、性別、感情 (笑顔など) などの顔の詳細を検出し、それらの詳細を表示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3

## EXIF およびその他のイメージ情報を保存する

次のコードサンプルは、以下の操作方法を示しています。

- JPG、JPEG、または PNG ファイルから EXIF 情報を取得します。
- Amazon S3 バケットにイメージファイルをアップロードします。
- Amazon Rekognition を使用して、ファイル内の 3 つの上位属性 (ラベル) を特定します。
- リージョンの Amazon DynamoDB テーブルに EXIF および ラベル情報を追加します。

### SDK Rust の

JPG、JPEG または PNG ファイルから EXIF 情報を取得し、イメージファイルを Amazon S3 バケットにアップロードし、Amazon Rekognition を使用してファイル内の 3 つの上位属性 (Amazon Rekognition のラベル) を識別し、EXIF および ラベル情報をリージョンの Amazon DynamoDB テーブルに追加します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3

## SDK for Rust を使用した Route 53 の例

次のコード例は、Route 53 で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### ListHostedZones

次のコード例は、ListHostedZones を使用する方法を示しています。

SDK Rust の

#### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。



```
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 詳細については、AWS SDK Rust API リファレンスの [ListHostedZones](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon S3 の例

次のコード例は、Amazon S3 で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1つのサービス内から、または他のAWSのサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

## 開始方法

### Hello Amazon S3

次のコード例は、Amazon S3の使用を開始する方法を示しています。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// 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.
/// * `filepath` - 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 utilities。

```

/// 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 詳細については、AWS SDK Rust API リファレンスの [List Buckets](#) 「」の「」を参照してください。

## トピック

- [基本](#)
- [アクション](#)
- [シナリオ](#)
- [サーバーレスサンプル](#)

## 基本

### 基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- バケットを作成し、そこにファイルをアップロードします。
- バケットからオブジェクトをダウンロードします。
- バケット内のサブフォルダにオブジェクトをコピーします。
- バケット内のオブジェクトを一覧表示します。
- バケットオブジェクトとバケットを削除します。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。



```

    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(())
}

```

シナリオで使用される一般的なアクション。

```

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 詳細については、AWS SDK Rust API リファレンスの以下のトピックを参照してください。
  - [CopyObject](#)
  - [CreateBucket](#)
  - [DeleteBucket](#)
  - [DeleteObjects](#)
  - [GetObject](#)

- [ListObjectsV2](#)
- [PutObject](#)

## アクション

### CompleteMultipartUpload

次の例は、CompleteMultipartUpload を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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 詳細については、AWS SDK Rust API リファレンスの [CompleteMultipartUpload](#) 「」の「」を参照してください。

## CopyObject

次のコード例は、CopyObject を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// 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 詳細については、AWS SDK Rust API リファレンスの [CopyObject](#) 「」の「」を参照してください。

## CreateBucket

次のコード例は、CreateBucket を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateBucket](#) 「」の「」を参照してください。

## CreateMultipartUpload

次のコード例は、CreateMultipartUpload を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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 詳細については、AWS SDK Rust API リファレンスの [CreateMultipartUpload](#) 「」の「」を参照してください。

## DeleteBucket

次の例は、DeleteBucket を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust APIリファレンスの[DeleteBucket](#)「」の「」を参照してください。

## DeleteObject

次の例は、DeleteObject を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// 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 詳細については、AWS SDK Rust APIリファレンスの[DeleteObject](#)「」の「」を参照してください。

## DeleteObjects

次の例は、DeleteObjects を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// 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 詳細については、AWS SDK Rust API リファレンスの [DeleteObjects](#) 「」の「」を参照してください。

## GetBucketLocation

次の例は、GetBucketLocation を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust APIリファレンスの[GetBucketLocation](#)「」の「」を参照してください。

## GetObject

次のコード例は、GetObject を使用する方法を示しています。

## SDK Rust の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GetObject](#) 「」の「」を参照してください。

## ListBuckets

次のコード例は、ListBuckets を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListBuckets](#) 「」の「」を参照してください。

## ListObjectVersions

次のコード例は、ListObjectVersions を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListObjectVersions](#) 「」の「」を参照してください。

## ListObjectsV2

次の例は、ListObjectsV2 を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK「Rust APIリファレンス」の「」の [ListObjectsV2](#)」を参照してください。

## PutObject

次のコード例は、PutObject を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

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 詳細については、AWS SDK Rust API リファレンスの [PutObject](#) 「」の「」を参照してください。

## UploadPart

次の例は、UploadPart を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [UploadPart](#) 「」の「」を参照してください。

## シナリオ

テキストを音声に変換し、テキストに戻す

次のコードサンプルは、以下の操作方法を示しています。

- Amazon Polly を使用して、プレーンテキスト (UTF-8) 入カファイルをオーディオファイルに合成します。
- Amazon S3 バケットに音声ファイルをアップロードします。
- Amazon Transcribe を使用して、音声ファイルをテキストに変換します。
- テキストを表示します。

### SDK Rust の

Amazon Polly を使用して、プレーンテキスト (UTF-8) 入カファイルをオーディオファイルに合成し、オーディオファイルを Amazon S3 バケットにアップロードし、Amazon Transcribe を使用してそのオーディオファイルをテキストに変換し、テキストを表示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- Amazon Polly
- Amazon S3
- Amazon Transcribe

### 署名付き を作成する URL

次のコード例は、Amazon S3 URL の署名付き を作成し、オブジェクトをアップロードする方法を示しています。

### SDK Rust の

#### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

## GET S3 オブジェクトへの事前署名リクエストを作成します。

```
/// 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 オブジェクトへの事前署名リクエストを作成します。

```
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())
}
```

## サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

### SDK Rust の

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

### イメージ内の顔を検出します

次のコードサンプルは、以下の操作方法を示しています。

- イメージを Amazon S3 バケットに保存します。
- Amazon Rekognition を使用して、年齢層、性別、感情 (笑顔など) などの顔の詳細を検出します。



- これらの詳細を表示します。

## SDK Rust の

アップロードプレフィックスを付け、Amazon S3 バケット内でイメージを保存し、Amazon Rekognition を使用して、年齢層、性別、感情 (笑顔など) などの顔の詳細を検出し、それらの詳細を表示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3

バケットが変更されている場合、バケットからオブジェクトを取得する

次のコード例は、S3 バケット内のオブジェクトからデータを読み取る方法を示しています。ただし、そのバケットが前回の取得時刻以降に変更されていない場合に限りです。

## SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GetObject](#) 「」の「」を参照してください。

## EXIF およびその他のイメージ情報を保存する

次のコードサンプルは、以下の操作方法を示しています。

- JPG、JPEG、または PNG ファイルから EXIF 情報を取得します。
- Amazon S3 バケットにイメージファイルをアップロードします。
- Amazon Rekognition を使用して、ファイル内の 3 つの上位属性 (ラベル) を特定します。

- リージョンの Amazon DynamoDB テーブルに EXIF および ラベル情報を追加します。

## SDK Rust の

JPG、JPEG、または PNG ファイルから EXIF 情報を取得し、イメージファイルを Amazon S3 バケットにアップロードし、Amazon Rekognition を使用してファイル内の 3 つの上位属性 (Amazon Rekognition のラベル) を識別し、EXIF および ラベル情報をリージョンの Amazon DynamoDB テーブルに追加します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3

## を使用したユニットテストと統合テスト SDK

次のコード例は、を使用してユニットテストと統合テストを記述する際のベストプラクティス手法の例を示しています AWS SDK。

## SDK Rust の

### Note

詳細については、「」を参照してください [GitHub](#)。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

Cargo.toml はテストサンプル用です。

```
[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"
```

オートモックとサービ斯拉ッパーを使ったユニットテストの例。

```
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);
}
}

```

を使用した統合テストの例 `StaticReplayClient`。

```

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",
        "astestsecretkey",
        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(&[]);
}

```

```
}  
}
```

## 大きなファイルをアップロードまたはダウンロードする

次のコード例は、Amazon S3 との間で大きなファイルをアップロードまたはダウンロードする方法を示しています。

詳細については、「[マルチパートアップロードを使用したオブジェクトのアップロード](#)」を参照してください。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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(())
}
```

## サーバーレスサンプル

Amazon S3 トリガーから Lambda 関数を呼び出す

次のコード例は、S3 バケットにオブジェクトをアップロードすることによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。関数は、イベントパラメータから S3 バケット名とオブジェクトキーを取得し、Amazon S3 API を呼び出してオブジェクトのコンテンツタイプを取得してログに記録します。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#) リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で S3 イベントを消費します。

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

## SageMaker SDK for Rust を使用した AI の例

次のコード例は、SageMaker AI で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

## アクション

### ListNotebookInstances

次の例は、ListNotebookInstances を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、[ListNotebookInstances](#) 「」のAWS SDK「Rust APIリファレンス」を参照してください。

## ListTrainingJobs

次の例は、ListTrainingJobs を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListTrainingJobs](#) 「」の「」を参照してください。

## SDK for Rust を使用した Secrets Manager の例

次のコード例は、Secrets Manager で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### トピック

## • [アクション](#)

# アクション

## GetSecretValue

次の例は、GetSecretValue を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの[GetSecretValue](#)「」の「」を参照してください。

## SDK for Rust SES API を使用した Amazon v2 の例

次のコード例は、Amazon v2 で for Rust SES API を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)
- [シナリオ](#)

## アクション

### CreateContact

次の例は、CreateContact を使用する方法を説明しています。

SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateContact](#) 「」の「」を参照してください。



## CreateContactList

次の例は、CreateContactList を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの[CreateContactList](#)「」の「」を参照してください。

## CreateEmailIdentity

次のコード例は、CreateEmailIdentity を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateEmailIdentity](#) 「」の「」を参照してください。

## CreateEmailTemplate

次の例は、CreateEmailTemplate を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateEmailTemplate](#) 「」の「」を参照してください。

## DeleteContactList

次の例は、DeleteContactList を使用する方法を説明しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DeleteContactList](#) 「」の「」を参照してください。

**DeleteEmailIdentity**

次の例は、DeleteEmailIdentity を使用する方法を説明しています。

## SDK Rust の

**Note**

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDKRust APIリファレンスの[DeleteEmailIdentity](#)「」の「」を参照してください。

## DeleteEmailTemplate

次の例は、DeleteEmailTemplate を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDKRust APIリファレンスの[DeleteEmailTemplate](#)「」の「」を参照してください。

## GetEmailIdentity

次のコード例は、GetEmailIdentity を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

E メールアドレスが検証されたかどうかを判定します。

```
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 詳細については、AWS SDK Rust API リファレンスの[GetEmailIdentity](#)「」の「」を参照してください。

## ListContactLists

次のコード例は、ListContactLists を使用する方法を示しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの[ListContactLists](#)「」の「」を参照してください。

## ListContacts

次の例は、ListContacts を使用する方法を説明しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListContacts](#) 「」の「」を参照してください。

## SendEmail

次のコード例は、SendEmail を使用する方法を示しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

連絡先リストのすべてのメンバーにメッセージを送信します。

```
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(())
}
```

テンプレートを使用して、連絡先リストのメンバー全員にメッセージを送信します。

```
    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 詳細については、AWS SDK Rust API リファレンスの [SendEmail](#) 「」の「」を参照してください。

## シナリオ

### ニュースレターワークフロー

次のコード例は、Amazon v2 SES API ニュースレターワークフローを実行する方法を示しています。

### SDK Rust の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの以下のトピックを参照してください。

- [CreateContact](#)

- [CreateContactList](#)
- [CreateEmailIdentity](#)
- [CreateEmailTemplate](#)
- [DeleteContactList](#)
- [DeleteEmailIdentity](#)
- [DeleteEmailTemplate](#)
- [ListContacts](#)
- [SendEmail.simple](#)
- [SendEmail.template](#)

## SDK for Rust を使用した Amazon SNS の例

次のコード例は、Amazon で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています SNS。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### トピック

- [アクション](#)
- [シナリオ](#)
- [サーバーレスサンプル](#)

## アクション

### CreateTopic

次のコード例は、CreateTopic を使用する方法を示しています。



## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [CreateTopic](#) 「」の「」を参照してください。

## ListTopics

次のコード例は、ListTopics を使用する方法を示しています。

## SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListTopics](#) 「」の「」を参照してください。

## Publish

次の例は、Publish を使用する方法を説明しています。

SDK Rust の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK「Rust APIリファレンス」の [発行](#) を参照してください。

## Subscribe

次の例は、Subscribe を使用する方法を説明しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

E メールアドレスをトピックにサブスクライブします。

```
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 詳細については、AWS SDK「Rust APIリファレンスの [サブスクリプト](#)」を参照してください。

## シナリオ

### サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

#### SDK Rust 用の

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## サーバーレスサンプル

Amazon SNSトリガーから Lambda 関数を呼び出す

次のコード例は、SNSトピックからメッセージを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用した Lambda でのSNSイベントの消費。

```
// 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 Rust を使用した Amazon SQS の例

次のコード例は、Amazon で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

### トピック

- [アクション](#)
- [サーバーレスサンプル](#)

## アクション

### ListQueues

次のコード例は、ListQueues を使用する方法を示しています。

SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

リージョンにリストされている最初の Amazon SQS キューを取得します。

```
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 詳細については、AWS SDK Rust API リファレンスの [ListQueues](#) 「」の「」を参照してください。

### ReceiveMessage

次のコード例は、ReceiveMessage を使用する方法を示しています。

SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [ReceiveMessage](#) 「」の「」を参照してください。

## SendMessage

次の例は、SendMessage を使用する方法を説明しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust APIリファレンスの[SendMessage](#)「」の「」を参照してください。

## サーバーレスサンプル

Amazon SQSトリガーから Lambda 関数を呼び出す

次のコード例は、SQSキューからメッセージを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用した Lambda でのSQSイベントの消費。

```
// 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トリガーを使用した Lambda 関数のバッチアイテム失敗のレポート

次のコード例は、SQSキューからイベントを受信する Lambda 関数に部分的なバッチレスポンスを実装する方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

### SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。 [サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda でSQSバッチアイテムの失敗を報告する。

```
// 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 Rust を使用した の例

次のコード例は、`aws_sdk_sts` で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています AWS STS。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

# アクション

## AssumeRole

次の例は、AssumeRole を使用する方法を説明しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [AssumeRole](#) 「」の「」を参照してください。

## SDK for Rust を使用した Systems Manager の例

次のコード例は、Systems Manager で for Rust を使用してアクションを実行し、AWS SDK一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [アクション](#)

### アクション

#### DescribeParameters

次のコード例は、DescribeParameters を使用する方法を示しています。

SDK Rust 用の

#### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [DescribeParameters](#) 「」の「」を参照してください。

## GetParameter

次のコード例は、GetParameter を使用する方法を示しています。

SDK Rust 用の

### Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [GetParameter](#) 「」の「」を参照してください。

## PutParameter

次のコード例は、PutParameter を使用する方法を示しています。

## SDK Rust 用の

 Note

詳細については、「」を参照してください GitHub。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 詳細については、AWS SDK Rust API リファレンスの [PutParameter](#) 「」の「」を参照してください。

## SDK for Rust を使用した Amazon Transcribe の例

次のコード例は、Amazon Transcribe で for Rust を使用してアクションを実行し、AWS SDK 一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には、完全なソースコードへのリンクが含まれており、そこからコンテキストにおけるコードの設定方法と実行方法についての手順を確認できます。

トピック

- [シナリオ](#)

## シナリオ

テキストを音声に変換し、テキストに戻す

次のコードサンプルは、以下の操作方法を示しています。

- Amazon Polly を使用して、プレーンテキスト (UTF-8) 入カファイルをオーディオファイルに合成します。
- Amazon S3 バケットに音声ファイルをアップロードします。
- Amazon Transcribe を使用して、音声ファイルをテキストに変換します。
- テキストを表示します。

SDK Rust 用の

Amazon Polly を使用して、プレーンテキスト (UTF-8) 入カファイルをオーディオファイルに合成し、オーディオファイルを Amazon S3 バケットにアップロードし、Amazon Transcribe を使用してそのオーディオファイルをテキストに変換し、テキストを表示します。

完全なソースコードと、セットアップと実行の手順については、「」の詳細な例を参照してください [GitHub](#)。

この例で使用されているサービス

- Amazon Polly
- Amazon S3
- Amazon Transcribe



# この AWS 製品またはサービスのセキュリティ

クラウドセキュリティは Amazon Web Services (AWS) の最優先事項です。AWS のお客様は、セキュリティを非常に重視する組織の要件を満たせるように構築されたデータセンターとネットワークアーキテクチャーから利点を得ます。セキュリティは、AWS とお客様の間での責任共有です。[責任共有モデル](#)では、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

クラウドのセキュリティ — AWS クラウドで提供されるすべてのサービスを実行するインフラストラクチャ AWS を保護し、安全に使用できるサービスを提供します。当社のセキュリティ責任は、最優先事項であり AWS、当社のセキュリティの有効性は、[AWS コンプライアンスプログラムの一環として、サードパーティーの監査者によって定期的にテストおよび検証されています](#)。

クラウド内のセキュリティ — お客様の責任は、使用している AWS サービス、およびお客様のデータの機密性、組織の要件、適用可能な法律や規制などのその他の要因によって決まります。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて[責任共有モデル](#)に従います。AWS サービスセキュリティ情報については、[AWS 「サービスセキュリティドキュメント」](#) ページと[AWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービス](#)を参照してください。

## トピック

- [この AWS 製品またはサービスのデータ保護](#)
- [この AWS 製品またはサービスのコンプライアンス検証](#)
- [この AWS 製品またはサービスのインフラストラクチャセキュリティ](#)
- [で最小TLSバージョンを適用する AWS SDK for Rust](#)

## この AWS 製品またはサービスのデータ保護

責任 [AWS 共有モデル](#)、この AWS 製品またはサービスのデータ保護に適用されます。このモデルで説明されているように、AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「AWS のサービス」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「[データプライバシーFAQ](#)」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログの[AWS 「責任共有モデル」とGDPR](#) ブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント、AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。1 TLS.2 が必要で、1.3 TLS をお勧めします。
- API とユーザーアクティビティのログ記録を でセットアップします AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「AWS CloudTrail ユーザーガイド」の [CloudTrail 「証跡の使用」](#) を参照してください。
- AWS 暗号化ソリューションと、その中のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは AWS を介して にアクセスするときに FIPS 140-3 検証済みの暗号化モジュールが必要な場合は API、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「[連邦情報処理規格 \(FIPS\) 140-3](#)」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、または を使用して、この AWS 製品またはサービスまたは他の AWS のサービス を操作する場合も同様です API AWS CLI AWS SDKs。タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。URL を外部サーバーに提供する場合は、そのサーバーへのリクエストを検証 URL するために認証情報を に含めないことを強くお勧めします。


## この AWS 製品またはサービスのコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「[コンプライアンス AWS のサービス プログラムによる対象範囲内コンプライアンス](#)」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、[AWS 「コンプライアンスプログラム」](#) を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「[Downloading AWS Artifact Reports](#)」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービスは、お客様のデータの機密性や貴社のコンプライアンス目的、適用される法律および規制によって決まります。は、コンプライアンスに役立つ以下のリソース AWS を提供します。

- [セキュリティのコンプライアンスとガバナンス](#) – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。
- [アマゾン ウェブ サービスHIPAAのセキュリティとコンプライアンスのためのアーキテクチャ](#) – このホワイトペーパーでは、企業が AWS を使用して HIPAA対象アプリケーションを作成する方法について説明します。

 Note

すべての AWS のサービスがHIPAA対象となるわけではありません。詳細については、[HIPAA「対象サービスリファレンス」](#)を参照してください。

- [AWS コンプライアンスリソース](#) – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- [AWS カスタマーコンプライアンスガイド](#) – コンプライアンスの観点から責任共有モデルを理解します。このガイドは、ガイダンスを保護し AWS のサービス、複数のフレームワーク (米国国立標準技術研究所 (NIST )、Payment Card Industry Security Standards Council (PCI )、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめたものです。
- [「デベロッパーガイド」の「ルールによるリソースの評価」](#) – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。AWS Config
- [AWS Security Hub](#) – これにより AWS のサービス、セキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「[Security Hub のコントロールリファレンス](#)」を参照してください。
- [Amazon GuardDuty](#) – これにより AWS アカウント、不審なアクティビティや悪意のあるアクティビティがないか環境を監視することで、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービスを検出します。PCI GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことでDSS、などのさまざまなコンプライアンス要件に対応するのに役立ちます。

- [AWS Audit Manager](#) – これにより AWS のサービス、AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて[責任共有モデル](#)に従います。AWS サービスセキュリティ情報については、[AWS 「サービスセキュリティドキュメント」](#) ページと[AWS、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービス](#)を参照してください。

## この AWS 製品またはサービスのインフラストラクチャセキュリティ

この AWS 製品またはサービスはマネージドサービスを使用するため、グローバルネットワークセキュリティによって保護されています AWS。AWS セキュリティサービスと [インフラストラクチャ AWS を保護する方法](#)については、[AWS 「クラウドセキュリティ」](#)を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱」の[「インフラストラクチャの保護」](#)を参照してください。 AWS

が AWS 公開したAPI呼び出しを使用して、ネットワーク経由でこの AWS 製品またはサービスにアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS )。1TLS.2 が必要で、1.3 TLS をお勧めします。
- ( エフェメラル Diffie-HellmanPFS) や DHE (エリプティック カーブ エフェメラル Diffie-Hellman) など、完全な前方秘匿性 ECDHE () を持つ暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットのアクセスキーを使用して署名する必要があります。または、[AWS Security Token Service](#) (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて[責任共有モデル](#)に従います。AWS サービスセキュリティ情報については、[AWS 「サービスセキュリティドキュメント」](#) ページと[AWS、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービス](#)を参照してください。

## で最小TLSバージョンを適用する AWS SDK for Rust

は AWS SDK for Rust を使用して TLS、AWS サービスと通信する際のセキュリティを強化します。は、デフォルトで 1.2 の最小 TLS バージョン SDK を適用します。デフォルトでは、は クライアントアプリケーションと サービスの両方で TLS 使用可能な の最高バージョン SDK もネゴシエートします。たとえば、は 1.3 TLS をネゴシエートできる SDK 場合があります。

が SDK 使用する TCP コネクタを手動で設定することで、アプリケーションに特定の TLS バージョンを適用できます。これを説明するために、次の例は 1.3 TLS を適用する方法を示しています。

### Note

一部の AWS サービスはまだ TLS 1.3 をサポートしていないため、このバージョンを適用すると SDK 相互運用性に影響する可能性があります。本番デプロイの前に、各サービスでこの設定をテストすることをお勧めします。

```
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/awslabs/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(())
}
```

## 付録: で使用される木箱のSDK鏘

この付録には、 で使用される木箱に関する詳細な情報が含まれています AWS SDK for Rust。これには、使用する Smithy コンポーネント、特定のビルド状況で使用する必要があるクレート、その他の情報が含まれます。

### スミティクレート

AWS SDK for Rust は、ほとんどのと同様に [Smithy](#) に基づいています AWS SDKs。Smithy は、 が提供するデータ型と関数を記述するために使用される言語です SDK。これらのモデルは、 SDK自体の構築に役立ちます。

SDK for Rust クレートのバージョンとその Smithy 依存関係のバージョンを確認する場合、これらのクレートはすべて [標準のセマンティックバージョン番号付け](#) を使用していることを知っておくと便利です。

Rust の Smithy クレートの詳細については、 [「Smithy Rust Design」](#) を参照してください。

### SDK for Rust で使用される木箱

によって発行された Smithy クレートは多数あります AWS。これらの中には、 Rust ユーザー SDK に関連するものもあれば、実装の詳細であるものもあります。

`aws-smithy-async`

非同期機能に Tokio を使用していない場合は、この木箱を含めます。

`aws-smithy-runtime`

すべての に必要な構成要素が含まれます AWS SDKs。

`aws-smithy-runtime-api`

で使用される基盤となるインターフェイス SDK。

`aws-smithy-types`

他の から再エクスポートされたタイプ AWS SDKs。複数の を使用する場合は、これを使用します SDKs。

`aws-smithy-types-convert`

に出入りするためのユーティリティ関数 `aws-smithy-types`。



## その他のクレート

次のクレートは存在しますが、それらについて何も知る必要はありません。

Rust ユーザーSDKにとって不要なサーバー関連のクレート :

- `aws-smithy-http-server`
- `aws-smithy-http-server-python`

SDK ユーザーが使用する必要がないコードを含む under-the-hood木箱 :

- `aws-smithy-checksum-callbacks`
- `aws-smithy-eventstream`
- `aws-smithy-http`
- `aws-smithy-protocol-test`
- `aws-smithy-query`
- `aws-smithy-json`
- `aws-smithy-xml`

サポートされておらず、将来使用できなくなる木箱 :

- `aws-smithy-client`
- `aws-smithy-http-auth`
- `aws-smithy-http-tower`



# ドキュメント履歴

このトピックでは、AWS SDK for Rust デベロッパーガイドの重要な変更点について説明します。

変更	説明	日付
<a href="#">目次の再編成</a>	設定と使用状況をより区別するために、目次を再編成します。	2024 年 7 月 1 日
<a href="#">の一般提供 AWS SDK for Rust</a>	新しいセキュリティ情報、新規および更新されたコード例、例を含むユニットテストに関する新しい詳細、の新しい一般提供リリースに関するその他の新規および更新されたコンテンツを含むようにガイドを更新しましたSDK。	2023 年 11 月 27 日
<a href="#">最小TLSバージョンの強制</a>	TLS でのバージョンを適用する方法に関する情報を追加しましたSDK。	2022 年 5 月 4 日
<a href="#">AWS SDK for Rust デベロッパープレビューリリース</a>	<a href="#">開発者プレビューリリース</a>	2021 年 12 月 2 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。