aws

开发人员指南

适用于 Rust 的 AWS SDK

适用于 Rust 的 AWS SDK: 开发人员指南

Table of Contents

适用于 Rust 的 AWS SDK 是什么？ 1
SDK 入门 1
SDK 主要版本的维护和支持 1
其他资源 1
入门 3
使用进行身份验证 AWS 3
更多身份验证信息 4
创建简单的应用程序 4
先决条件 4
创建第一个 SDK 应用程序 5
基础知识 6
先决条件 4
Rust 基础知识 7
适用于 Rust 的 AWS SDK 箱子基础知识 8
用于使用的项目配置 AWS 服务 8
Tokio 运行时 9
配置服务客户端 10
在外部配置客户端 11
代码中的客户端配置 12
从环境中配置客户端 12
使用生成器模式进行特定于服务的设置 13
高级显式客户端配置 14
AWS 区域 14
AWS 区域 提供者链 14
设置输入 AWS 区域 代码 15
凭证提供程序 16
凭证提供程序链 17
显式凭证提供程序 19
身份缓存 19
行为版本 19
在 Cargo.toml 中设置行为版本 20
在代码中设置行为版本 20
重试 20
默认重试配置 21

最大尝试次数 21
延迟和回退 22
自适应重试模式 22
超时 23
API 超时 23
停滞流保护 24
可观测性 25
日志记录 25
客户端端点 29
自定义配置 29
示例 32
覆盖操作配置 33
HTTP 35
选择替代 TLS 提供程序 35
启用 FIPS 支持 37
优先考虑后量子密钥交换 38
覆盖 DNS 解析器 38
自定义根 CA 证书 39
拦截器 40
拦截器注册 41
使用 SDK 43
发出服务请求 43
最佳实践 44
尽可能重复使用 SDK 客户端 44
配置 API 超时 45
并发 45
术语 45
一个简单示例 45
所有权和可变性 47
更多术语！ 47
重写我们的示例以提高效率（单线程并发） 48
重写我们的示例以提高效率（多线程并发） 50
调试多线程应用程序 52
创建 Lambda 函数 52
创建预签名 URL 52
预签名基础知识 53

预签名 POST 和 PUT 请求 54
独立签署人 54
处理错误 56
服务错误 56
错误元数据 57
使用 DisplayErrorContext 打印详细的错误 57
分页 59
单元测试 61
使用 mockall 进行单元测试 61
静态重播 66
使用 aws-smithy-mocks 进行单元测试 70
Waiter 76
代码示例 78
API Gateway 79
操作 80
场景 81
AWS 社区捐款 81
API Gateway 管理 API 82
操作 80
Application Auto Scaling 83
操作 80
Aurora 84
开始使用 85
基本功能 86
操作 80
Auto Scaling 232
开始使用 85
基本功能 86
操作 80
Amazon Bedrock 运行时系统 266
场景 81
Anthropic Claude 276
Amazon Bedrock 代理运行时 291
操作 80
Amazon Cognito 身份提供者 296
操作 80

Amazon Cognito Sync 297
操作 80
Firehose 299
操作 80
Amazon DocumentDB 300
无服务器示例 300
DynamoDB 302
操作 80
场景 81
无服务器示例 300
AWS 社区捐款 81
Amazon EBS 319
操作 80
Amazon EC2 322
开始使用 85
基本功能 86
操作 80
Amazon ECR 383
操作 80
Amazon ECS 386
操作 80
Amazon EKS 388
操作 80
AWS Glue 390
开始使用 85
基本功能 86
操作 80
IAM 405
开始使用 85
基本功能 86
操作 80
AWS IoT 432
操作 80
Kinesis 434
操作 80
无服务器示例 300

AWS KMS 442
操作 80
Lambda 450
基本功能 86
操作 80
场景 81
无服务器示例 300
AWS 社区捐款 81
MediaLive 500
操作 80
MediaPackage 501
操作 80
Amazon MSK 503
无服务器示例 300
Amazon Polly 505
操作 80
场景 81
Amazon RDS 510
无服务器示例 300
Amazon RDS 数据服务 514
操作 80
Amazon Rekognition 515
场景 81
Route 53 517
操作 80
Amazon S3 518
开始使用 85
基本功能 86
操作 80
场景 81
无服务器示例 300
SageMaker AI 568
操作 80
Secrets Manager 571
操作 80
Amazon SES API v2 572

操作 80
场景 81
Amazon SNS 588
操作 80
场景 81
无服务器示例 300
Amazon SQS 594
操作 80
无服务器示例 300
AWS STS 599
操作 80
Systems Manager 600
操作 80
Amazon Transcribe 603
场景 81
安全性 605
数据保护 605
合规性验证 606
基础设施安全性 606
强制使用最低版本的 TLS 607
SDK 使用的 crate 609
Smithy crate 609
可与 SDK 结合使用的 crate 609
其他 crate 609
文档历史记录 611
..... dcxii

适用于 Rust 的 AWS SDK 是什么？

Rust 是一种没有垃圾收集器的系统编程语言，专注于三个目标：安全性、速度和并发性。

适用于 Rust 的 AWS SDK 提供用于与 AWS 基础设施服务进行交互的 Rust API。利用此 SDK，您可以基于 Amazon S3、Amazon EC2 和 DynamoDB 等构建应用程序。

主题

- SDK 入门
- SDK 主要版本的维护和支持
- 其他资源

SDK 入门

如果您是首次接触 SDK 的用户，建议您先阅读适用于 Rust 的 SDK 入门。

有关配置和设置，包括如何创建和配置用于向 AWS 服务发出请求的服务客户端，请参阅在适用于 Rust 的 AWS SDK 中配置服务客户端。

有关使用 SDK 的信息，请参阅使用适用于 Rust 的 AWS SDK。

有关此 Rust 代码示例的完整列表，请参阅代码示例。

SDK 主要版本的维护和支持

有关维护和支持 SDK 主要版本及其基础依赖关系的信息，请参阅 AWS SDK 和工具参考指南中的以下内容：

- AWS SDK 和工具维护策略
- AWS SDKs and Tools Version Support Matrix

其他资源

除了本指南外，还有以下适用于 SDK 开发人员的有价值的在线资源：

- AWS SDK 和工具参考指南：包含 AWS SDK 中常见的设置、功能和其他基础概念。

- Rust 编程语言网站
- 适用于 Rust 的 AWS SDKAPI 参考
- 适用于 Rust 的 AWS SDK 的 AWS 开发人员工具博客
- GitHub 上的适用于 Rust 的 AWS SDK 源代码
- 适用于 Rust 的 AWS SDK 的 AWS 代码示例目录

适用于 Rust 的 SDK 入门

了解如何安装、设置和使用 SDK 创建 Rust 应用程序，以便以编程方式访问 AWS 资源。

主题

- AWS使用AWS适用于 Rust 的 SDK 进行身份验证
- 使用适用于 Rust 的 AWS SDK 创建简单的应用程序
- 的基础知识 适用于 Rust 的 AWS SDK

AWS使用AWS适用于 Rust 的 SDK 进行身份验证

使用开发AWS时，您必须确定您的代码是如何进行身份验证的。AWS 服务您可以根据环境和可用的访问权限以不同的方式配置对AWS资源的编程AWS访问权限。

要选择您的身份验证方法并针对 SDK 进行配置，请参阅和工具参考指南中的身份验证AWSSDKs 和访问。

我们建议在本地开发且雇主未向其提供身份验证方法的新用户进行设置AWS IAM Identity Center。此方法包括安装AWS CLI以便于配置和定期登录AWS访问门户。

如果您选择此方法，请完成AWSSDKs 和工具参考指南中的使用控制台凭据登录进行AWS本地开发的过程。此后，您的环境应包含以下元素：

- AWS CLI，用于在运行应用程序之前启动AWS访问门户会话。
- 共享 AWSconfig 文件，其 [default] 配置文件包含一组可从 SDK 中引用的配置值。要查找此文件的位置，请参阅AWSSDKs 和工具参考指南中的共享文件的位置。
- 共享 config 文件设置了 region 设置。这将设置 SDK 用于AWS请求的默认值AWS 区域。此区域用于未指定使用区域的 SDK 服务请求。
- 在向发送请求之前，SDK 使用配置文件的登录凭据提供程序配置来获取凭据。AWS 该login_session值存储您在登录工作流程中选择的管理控制台会话的身份，允许访问应用程序中使用的AWS服务。

  以下示例config文件显示了在登录工作流程中选择登录凭据提供程序配置控制台会话时设置的默认配置文件。配置文件的login_session设置是指在工作流程期间选择的指定控制台会话：

```
[default]
```

```
login_session = arn:aws:iam::0123456789012:user/username
region = us-east-1
```

> **Note**
>
> 您必须启用 aws-config crate 的credentials-login功能才能使用此凭证提供商。

更多身份验证信息

人类用户，也称为人类身份，是应用程序的人员、管理员、开发人员、操作员和使用者。他们必须具有身份才能访问您的AWS环境和应用程序。作为组织成员的人类用户（即您、开发人员）也称为工作人员身份。

访问时使用临时证书AWS。您可以为人类用户使用身份提供商，通过扮演提供临时证书的角色来提供对AWS账户的联合访问权限。对于集中式访问权限管理，我们建议使用 AWS IAM Identity Center (IAM Identity Center) 来管理对您账户的访问权限以及这些账户中的其他权限。有关更多替代方案，请参阅以下内容：

- 有关最佳实践的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。
- 要创建短期AWS证书，请参阅 IAM 用户指南中的临时安全证书。
- 要了解 SDK for Rust 支持的其他凭证提供商，请参阅AWSSDKs 和工具参考指南中的标准化凭证提供商。

使用适用于 Rust 的 AWS SDK 创建简单的应用程序

您可以按照本教程创建调用 Rust 的简单应用程序，从而快速开始使用 AWS SDK for Rust AWS 服务。

先决条件

要使用，必须安装 Rust 和 Cargo。适用于 Rust 的 AWS SDK

- 安装 Rust 工具链：https://www.rust-lang。 org/tools/install
- 通过运行以下命令安装 cargo-component 工具：cargo install cargo-component

推荐的工具：

可以在 IDE 中安装以下可选工具，以帮助完成代码和进行故障排除。

- 有关 rust-analyzer 扩展程序，请参阅 Visual Studio Code 中的 Rust。
- 有关 Amazon Q 开发者版，请参阅在 IDE 中安装 Amazon Q 开发者版扩展程序或插件。

创建第一个 SDK 应用程序

此过程创建了您的第一个适用于 Rust 的 SDK 应用程序，其中列出了您的 DynamoDB 表。

1. 在终端或控制台窗口中，导航到计算机上要创建应用程序的位置。
2. 输入以下命令以创建 `hello_world` 目录并在其中填充一个框架 Rust 项目：

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

3. 导航到 `hello_world` 目录并使用以下命令向应用程序添加所需的依赖项：

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

这些依赖项包括为 DynamoDB 提供配置功能和支持的 SDK crate，其中包括用于实现异步 I/O 操作的 `tokio` crate。

> **Note**
>
> 除非您使用像 `tokio/full` 这样的功能，否则 Tokio 不会提供异步运行时。适用于 Rust 的 SDK 需要异步运行时。
> 该`aws-config/credentials-login`功能支持AWS管理控制台登录凭据，有关更多信息，请参阅《工具参考指南》AWSSDKs 和《工具参考指南》中的身份验证和访问权限。

4. 在 `src` 目录中更新 `main.rs` 以包含以下代码。

```
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**
>
> 此示例只显示第一页结果。要了解如何处理多页结果，请参阅the section called “分页”。

5. 运行程序：

```
$ cargo run
```

您应该会看到表名称列表。

的基础知识 适用于 Rust 的 AWS SDK

学习编程基础知识 适用于 Rust 的 AWS SDK，例如：Rust 编程语言基础知识、有关 Rust 板条箱的 SDK 的信息、项目配置以及适用于 Rust 使用 Tokio 运行时的 SDK。

先决条件

要使用，必须安装 Rust 和 Cargo。 适用于 Rust 的 AWS SDK

- 安装 Rust 工具链：https://www.rust-lang。 org/tools/install
- 通过运行以下命令安装 `cargo-component` 工具：`cargo install cargo-component`

推荐的工具：

可以在 IDE 中安装以下可选工具，以帮助完成代码和进行故障排除。

- 有关 rust-analyzer 扩展程序，请参阅 Visual Studio Code 中的 Rust。
- 有关 Amazon Q 开发者版，请参阅在 IDE 中安装 Amazon Q 开发者版扩展程序或插件。

Rust 基础知识

以下是 Rust 编程语言的一些基础知识，了解这些基础知识会很有帮助。有关更多信息的所有参考资料均来自 Rust 编程语言。

- `Cargo.toml` 是标准 Rust 项目配置文件，它包含该项目的依赖项和一些元数据。Rust 源文件具有 `.rs` 文件扩展名。请参阅 Hello, Cargo!。
  - 您可以使用配置文件自定义 `Cargo.toml`，请参阅使用版本配置文件自定义构建。这些配置文件完全无关，与共享 AWS `config`文件中 AWS配置文件的使用无关。
  - 向项目和此文件添加库依赖项的常用方法是使用 `cargo add`。请参阅`cargo-add`。
- Rust 有一个基本函数结构，如下所示。`let` 关键字声明了变量，可以与赋值（`=`）结合使用。如果您没有在 `let` 之后指定类型，编译器会推断出类型。请参阅变量和可变性。

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

- 要声明变量 `x` 具有显式类型 `T`，Rust 使用语法 `x: T`。请参阅数据类型。
- `struct X {}` 定义新类型 `X`。方法是在自定义结构类型 `X` 上实现的。类型 `X` 的方法是用以关键字 `impl` 为前缀的实现块声明的。在实现块中，`self` 是指调用该方法的结构的实例。请参阅关键字 `impl` 和方法语法。

- 如果一个感叹号（“!”）跟在看似函数定义或函数调用的后面，则表示代码正在定义或调用宏。请参阅宏。
- 在 Rust 中，不可恢复的错误由 `panic!` 宏表示。当程序遇到 `panic!` 时，它会停止运行，打印失败消息，展开，清理堆栈，然后退出。请参阅 panic！不可恢复的错误。
- 与其他编程语言不同，Rust 不支持从基类继承功能；Rust 通过 `traits` 提供方法重载。特性在概念上可视为类似于接口。但是，特性和真实接口存在差异，并且在设计过程中通常以不同的方式使用。请参阅特性：定义共享行为。
  - 多态性是指代码支持多种数据类型的功能，而无需单独为每种数据类型编写代码。Rust 通过枚举、特性和泛型支持多态性。请参阅继承作为类型系统和代码共享。
- Rust 在内存方面非常显式。智能指针“是像指针一样起作用的数据结构，但也具有额外的元数据和功能”。请参阅智能指针。
  - 该类型Cow是一个 clone-on-write智能指针，可帮助在必要时将内存所有权转移给调用者。请参阅`Enum std::borrow::Cow`。
  - 类型 `Arc` 是用于计算已分配实例的原子引用计数智能指针。请参阅`Struct std::sync::Arc`。
- 适用于 Rust 的 SDK 经常使用生成器模式来构造复杂类型。

适用于 Rust 的 AWS SDK 箱子基础知识

- 适用于 Rust 的 SDK 功能的主要核心 crate 是 `aws-config`。它之所以包含在大多数项目中，是因为它提供了从环境中读取配置的功能。

```
$ cargo add aws-config
```

  - 不要将其与所谓 AWS 服务 AWS Config的混淆。由于这是一项服务，因此它遵循 AWS 服务 板条箱的标准惯例并被调`aws-sdk-config`用。
- 适用于 Rust 的 SDK 库被每个 AWS 服务库分成不同的库箱。这些 crate 可通过 https://docs.rs/ 获取。
- AWS 服务 板条箱遵循的命名惯例`aws-sdk-`*`[servicename]`*，例如`aws-sdk-s3`和。`aws-sdk-dynamodb`

用于使用的项目配置 AWS 服务

- 你需要在项目中为每个 AWS 服务 你想让你的应用程序使用的箱子添加一个 crate。

- 添加 crate 的推荐方法是在项目目录中使用命令行运行 `cargo add [crateName]`，例如 `cargo add aws-sdk-s3`。
  - 这将向项目 `Cargo.toml` 中的 `[dependencies]` 下添加一行。
  - 默认情况下，这会将最新版本的 crate 添加到项目中。
- 在源文件中，使用 `use` 语句将 crate 中的项目放入作用域。请参阅 Rust 编程语言网站上的使用外部程序包。
  - crate 名称通常使用连字符，但是在实际使用 crate 时，连字符会被转换为下划线。例如，`aws-config` crate 在代码 `use` 语句中使用，例如：`use aws_config`。
- 配置是一个复杂的主题。配置可以直接在代码中进行，也可以在外部环境变量或配置文件中指定。有关更多信息，请参阅 在外部配置适用于 Rust 的 AWS SDK 服务客户端。
  - 当 SDK 加载配置时，系统会记录无效值，而不是停止执行，因为大多数设置都有合理的默认值。要了解如何启用日志记录功能，请参阅在适用于 Rust 的 AWS SDK 中配置和使用日志记录。
  - 大多数环境变量和配置文件设置会在程序启动时加载一次。在重新启动程序之前，不会看到对这些值的任何更新。

Tokio 运行时

- Tokio 是适用于 Rust 的 SDK 编程语言的一个异步运行时，它执行 `async` 任务。请参阅 tokio.rs 和 docs.rs/tokio。
- 适用于 Rust 的 SDK 需要异步运行时。建议您在项目中添加以下 crate：

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

- `tokio::main` 属性宏为程序创建一个异步主入口点。要使用此宏，请将其添加到 `main` 方法前面的行中，如下所示：

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

在适用于 Rust 的 AWS SDK 中配置服务客户端

要以编程方式访问 AWS 服务，适用于 Rust 的 AWS SDK 对每个 AWS 服务使用一个客户端结构体。例如，如果您的应用程序需要访问 Amazon EC2，则您的应用程序会创建一个 Amazon EC2 客户端结构体来与该服务交互。然后，您可以使用服务客户端向该 AWS 服务 发出请求。

要向 AWS 服务发出请求，您必须先创建和配置服务客户端。对于您的代码使用的每个 AWS 服务，它都有自己的 crate 和用于与之交互的专用类型。客户端为服务公开的每个 API 操作公开一种方法。

配置 SDK 行为的方法有很多，但归根结底，一切都与服务客户端的行为有关。除非使用基于配置创建的服务客户端，否则任何配置都不会生效。

在使用 AWS 服务进行开发时，您必须确定您的代码是如何使用 AWS 进行身份验证的。您还必须设置要使用的 AWS 区域。

AWS SDK 和工具参考指南还介绍了在许多 AWS SDK 中常见的设置、功能和其他基础概念。

主题

- 在外部配置适用于 Rust 的 AWS SDK 服务客户端
- 在代码中为 Rust 服务客户端配置 AWS SDK
- AWS 区域 为 适用于 Rust 的 AWS SDK
- 使用 AWS 适用于 Rust 凭证提供商的 SDK
- 在中使用行为版本 适用于 Rust 的 AWS SDK
- 在 Rust AWS 开发工具包中配置重试次数
- 在 Rust AWS 开发工具包中配置超时
- 在适用于 Rust 的 AWS SDK 中配置可观测性功能
- 在中配置客户端终端节点 适用于 Rust 的 AWS SDK
- 在适用于 Rust 的 AWS SDK 中覆盖客户端的单个操作配置
- 在 Rust AWS 开发工具包中配置 HTTP 级别的设置
- 在 Rust AWS 开发工具包中配置拦截器

在外部配置适用于 Rust 的 AWS SDK 服务客户端

许多配置设置可以通过代码以外的方式来处理。在外部处理配置时，配置将应用于您的所有应用程序。大多数配置设置可以设置为环境变量，也可以在单独的共享 AWS `config` 文件中设置。共享 `config` 文件可以维护多组独立的设置（称为配置文件），以便为不同的环境或测试提供不同的配置。

环境变量和共享 `config` 文件设置都已标准化，可在 AWS SDK 和工具之间共享，从而支持在不同编程语言和应用程序之间保持一致的功能。

请参阅《AWS SDK 和工具参考指南》，了解如何通过这些方法配置应用程序，以及每个跨 SDK 设置的详细信息。要查看 SDK 可以从环境变量或配置文件中解析的所有设置，请参阅《AWS SDK 和工具参考指南》中的设置参考。

要向 AWS 服务发出请求，请先实例化该服务对应的客户端。您可以为服务客户端配置常用设置，例如超时、HTTP 客户端及重试配置。

每个服务客户端都需要一个 AWS 区域和一个凭证提供程序。SDK 使用这些值将您的资源请求发送到正确的区域，并使用正确的凭证对请求进行签名。您可以在代码中以编程方式指定这些值，也可以让它们从环境变量中自动加载。

该 SDK 有一系列地点（或来源），它会检查这些地点（或来源），以便找到配置设置的值。

1. 在代码中或服务客户端本身上设置的任何显式设置均优先于其他任何设置。
2. 环境变量
   - 有关设置环境变量的详细信息，请参阅《AWS SDK 和工具参考指南》中的环境变量。
   - 请注意，您可以在不同作用域层级为 shell 配置环境变量：系统级、用户级，以及特定终端会话级。
3. 共享`config`文件和`credentials`文件
   - 有关设置这些文件的详细信息，请参阅《AWS SDK 和工具参考指南》中的共享 `config` 和 `credentials` 文件。
4. 最后才会使用 SDK 源代码本身提供的任何默认值。
   - 某些属性（例如“区域”）没有默认值。您必须在代码、环境设置或共享 `config` 文件中明确指定这些属性。如果 SDK 无法解析所需的配置，API 请求在运行时会失败。

在代码中为 Rust 服务客户端配置 AWS SDK

当直接在代码中处理配置时，配置范围仅限于使用该代码的应用程序。在该应用程序中，您可选择以下配置方式：对所有服务客户端的全局配置、对某一特定 AWS 服务 类型所有客户端的配置，或对某个具体服务客户端实例的配置。

要向发出请求 AWS 服务，请先为该服务实例化客户端。您可以为服务客户端配置常用设置，例如超时、HTTP 客户端及重试配置。

每个服务客户端都需要一个 AWS 区域 和一个凭据提供商。SDK 使用这些值将您的资源请求发送到正确的区域，并使用正确的凭证对请求进行签名。您可以在代码中以编程方式指定这些值，也可以让它们从环境变量中自动加载。

> **Note**
>
> 服务客户端的构造成本可能很高，而且通常需要共享。为了便于实现这一点，所有 `Client` 结构体均实施 `Clone`。

从环境中配置客户端

要创建具有环境源配置的客户端，请使用 `aws-config` crate 中的静态方法：

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

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

在 Amazon Elastic Compute Cloud 上运行时 AWS Lambda，或者在任何其他可以直接从环境中配置服务客户端的环境中运行时，以这种方式创建客户端非常有用。这样可以将您的代码与其运行的环境分离，并且可以更轻松地将您的应用程序部署到多个环境中， AWS 区域 而无需更改代码。

您可以显式覆盖特定属性。显式配置优先于从执行环境中解析的配置。以下示例从环境加载配置，但显式覆盖 AWS 区域：

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

```
    .load()
    .await;

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

> **Note**
>
> 并非所有配置值都是在创建时由客户端提供的。当使用客户端发出请求时，凭证提供程序层会访问与凭证相关的设置，例如临时访问密钥和 IAM Identity Center 配置。

前面示例中显示的代码 `BehaviorVersion::latest()` 指出了默认使用的 SDK 版本。`BehaviorVersion::latest()` 适用于大多数情况。有关更多信息，请参阅 在中使用行为版本适用于 Rust 的 AWS SDK。

使用生成器模式进行特定于服务的设置

有些选项只能在特定的服务客户端类型上配置。但是，大多数情况下，您仍然需要从环境中加载大部分配置，然后专门添加其他选项。建造者模式是 适用于 Rust 的 AWS SDK 箱子里常见的模式。您首先使用 `aws_config::defaults` 加载常规配置，然后使用 `from` 方法将该配置加载到您正在使用的服务的生成器中。然后，您可以为该服务设置任何唯一的配置值并调用 `build`。最后，根据此修改后的配置创建客户端。

```
// 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);
```

要发现特定类型服务客户端可用的其他方法，一种方法是使用 API 文档，例如 `aws_sdk_s3::config::Builder` 的文档。

高级显式客户端配置

要使用特定值配置服务客户端，而不是从环境中加载配置，您可以在客户端 Config 生成器上指定这些值，如下所示：

```
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);
```

使用 aws_sdk_s3::Config::builder() 创建服务配置时，不会加载任何默认配置。只有在基于 aws_config::defaults 创建配置时才会加载默认配置。

有些选项只能在特定的服务客户端类型上配置。前面的示例显示了如何在 Amazon S3 客户端上使用 endpoint_resolver 函数来实现这一点。

AWS 区域 为 适用于 Rust 的 AWS SDK

您可以使用访问 AWS 服务 在特定地理区域运营的内容 AWS 区域。它可以用于保证冗余，并保证您的数据和应用程序接近您和用户访问它们的位置。有关如何使用区域的更多信息，请参阅AWS SDKs 和工具参考指南AWS 区域中的。

> **Important**
>
> 大多数资源都位于特定的区域 AWS 区域 ，在使用 SDK 时，您必须为该资源提供正确的区域。

你必须为 Rust AWS 区域 的 SDK 设置默认值才能用于 AWS 请求。此默认设置用于未指定区域的任何 SDK 服务方法调用。

有关如何通过共享 AWS config文件或环境变量设置默认区域的示例，请参阅AWS SDKs 和工具参考指南AWS 区域中的。

AWS 区域 提供者链

从执行环境加载服务客户端的配置时，将使用以下查找过程。SDK 找到的第一个已设置的值将用于客户端配置。有关创建服务客户端的更多信息，请参阅从环境中配置客户端。

1. 以编程方式设置显式区域。
2. 系统会检查 AWS_REGION 环境变量。
   - 如果您正在使用该 AWS Lambda 服务，则此环境变量由 AWS Lambda 容器自动设置。
3. 已选中共享 AWS config文件中的region属性。
   - AWS_CONFIG_FILE 环境变量可用于更改共享 config 文件的位置。要详细了解此文件的保存位置，请参阅《工具参考指南》config和《工具参考指南》中的共享credentials文件AWSSDKs 和文件的位置。
   - AWS_PROFILE 环境变量可用于选择命名配置文件而不是默认配置文件。要了解有关配置不同配置文件的更多信息，请参阅config和工具参考指南中的共享AWS SDKs 和*credentials*文件。
4. SDK 将尝试使用 Amazon EC2 实例元数据服务，为当前运行的 Amazon EC2 实例确定区域。
   - 适用于 Rust 的 AWS SDK 唯一的支持 IMDSv2。

在创建与服务客户端一起使用的基本配置时，系统会自动使用 RegionProviderChain，无需额外的代码：

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

设置输入 AWS 区域 代码

在代码中显式设置区域

如果要显式设置区域，请直接在配置中使用 Region::new()。

未使用区域提供程序链，不检查环境、共享 config 文件或 Amazon EC2 实例元数据服务。

```
use aws_config::{defaults, BehaviorVersion};
use aws_sdk_s3::config::Region;

#[tokio::main]
async fn main() {
    let config = defaults(BehaviorVersion::latest())
        .region(Region::new("us-west-2"))
        .load()
        .await;

    println!("Using Region: {}", config.region().unwrap());
```

```
}
```

请务必为输入有效的字符串 AWS 区域；所提供的值未经过验证。

自定义 `RegionProviderChain`

如果您想有条件地注入区域、覆盖它或自定义解析链，请使用AWS 区域 提供者链。

```
use aws_config::{defaults, BehaviorVersion};
use aws_config::meta::region::RegionProviderChain;
use aws_sdk_s3::config::Region;
use std::env;

#[tokio::main]
async fn main() {
    let region_provider =
 RegionProviderChain::first_try(env::var("CUSTOM_REGION").ok().map(Region::new))
        .or_default_provider()
        .or_else(Region::new("us-east-2"));

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

    println!("Using Region: {}", config.region().unwrap());
}
```

之前的配置将：

1. 首先检查 CUSTOM_REGION 环境变量中是否设置了字符串。
2. 如果不可用，请回退到默认的区域提供程序链。
3. 如果失败了，请使用“us-east-2”作为最终回退选项。

使用 AWS 适用于 Rust 凭证提供商的 SDK

对的所有请求都 AWS 必须使用颁发的凭证进行加密签名。 AWS运行时，SDK 会通过检查多个位置来获取凭证的配置值。

如果检索到的配置包含 AWS IAM Identity Center 单点登录访问设置，则 SDK 将与 IAM Identity Center 配合检索用于向 AWS 服务发出请求的临时凭证。

如果检索到的配置包含临时证书，SDK 将使用它们进行 AWS 服务 调用。临时凭证由访问密钥对和会话令牌组成。

身份验证 AWS 可以在您的代码库之外处理。SDK 可通过凭证提供程序链自动检测、使用并刷新多种身份验证方式。

有关项目 AWS 身份验证入门的指导选项，请参阅和工具参考指南中的身份验证AWS SDKs 和访问权限。

凭证提供程序链

如果您在构建客户端时没有显式指定凭证提供程序，那么适用于 Rust 的 SDK 会使用凭证提供程序链来检查一系列可以提供凭证的位置。SDK 在其中一个位置找到凭证后，搜索就会停止。有关构建客户端的详细信息，请参阅在代码中为 Rust 服务客户端配置 AWS SDK。

以下示例未在代码中指定凭证提供程序。SDK 使用凭证提供程序链来检测在托管环境中设置的身份验证，并将该身份验证用于 AWS 服务调用。

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

凭证检索顺序

凭证提供程序链使用以下预定义序列搜索凭证：

1. 访问密钥环境变量

   SDK 尝试从 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 以及 `AWS_SESSION_TOKEN` 环境变量加载凭证。

2. 共享的 AWS **`config`**和**`credentials`**文件

   SDK 尝试从共享 AWS `config`和`credentials`文件中的`[default]`配置文件加载凭证。您可以使用 `AWS_PROFILE` 环境变量来选择您想让 SDK 加载的命名配置文件，而不是使用`[default]`。`config`和`credentials`文件由各种 AWS SDKs 工具共享。有关这些文件的更多信息，请参阅`config`和工具参考指南中的共享AWS SDKs 和*`credentials`*文件。有关可在配置文件中指定的标准化提供商的更多信息，请参阅AWS SDKs 和工具标准化凭证提供商。

3. AWS STS 网络身份

在创建需要访问的移动应用程序或基于客户端的 Web 应用程序时 AWS， AWS Security Token Service (AWS STS) 会为通过公共身份提供商 (IdP) 进行身份验证的联合用户返回一组临时安全证书。

- 当您在配置文件中指定此项时，SDK 或工具会尝试使用 AWS STS `AssumeRoleWithWebIdentity` API 方法检索临时证书。有关此方法的详细信息，请参阅 AWS Security Token Service API 参考AssumeRoleWithWebIdentity中的。
- 有关配置此提供程序的指导，请参阅《和工具参考指南》AWS SDKs 中的 “使用网络身份进行联合” 或 OpenID Connect。
- 有关此提供商的 SDK 配置属性的详细信息，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的代入角色凭据提供程序。

4. Amazon ECS 和 Amazon EKS 容器凭证

   您的 Amazon Elastic Container Service 任务和 Kubernetes 服务账户可以具有与其关联的 IAM 角色。在 IAM 角色中授予的权限由任务中运行的容器或容器组（Pod）内的容器承担。此角色允许适用于 Rust 的 SDK 应用程序代码（在容器上）使用其他 AWS 服务。

   SDK 尝试从 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` 或 `AWS_CONTAINER_CREDENTIALS_FULL_URI` 环境变量检索凭证，这些变量可以由 Amazon ECS 和 Amazon EKS 自动设置。

   - 有关为 Amazon ECS 设置此角色的详细信息，请参阅《Amazon Elastic Container Service 开发人员指南》中的 Amazon ECS 任务 IAM 角色。
   - 有关 Amazon EKS 的设置信息，请参阅《Amazon EKS 用户指南》中的设置 Amazon EKS 容器组身份代理。
   - 有关此提供商的 SDK 配置属性的详细信息，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的容器凭证提供程序。

5. Amazon EC2 实例元数据服务

   创建 IAM 角色并将其附加到您的实例。实例上适用于 Rust 的 SDK 应用程序会尝试从实例元数据中检索由角色提供的凭证。

   - 适用于 Rust 的软件开发工具包仅支持IMDSv2。
   - 有关设置此角色和使用元数据的详细信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 的 IAM 角色和使用实例元数据。
   - 有关此提供商的 SDK 配置属性的详细信息，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的 IMDS 凭证提供程序。

6. 如果此时仍未解析凭证，则操作 panics 会出现错误。

有关 AWS 凭据提供程序配置设置的详细信息，请参阅《工具参考指南》的 “设置” 参考中的标准化凭据提供程序。AWS SDKs

显式凭证提供程序

您可以指定 SDK 应使用的特定凭证提供程序，而不是依赖凭证提供程序链来检测您的身份验证方法。使用 `aws_config::defaults` 加载常规配置时，您可以指定自定义凭证提供程序，如下所示：

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

您可以通过实现 `ProvideCredentials` 特性来实施自己的凭证提供程序。

身份缓存

SDK 将缓存凭证和其他身份类型，例如 SSO 令牌。默认情况下，SDK 使用惰性缓存实现：在首次请求时加载凭证并缓存，当凭证即将过期时，会在下次请求时尝试刷新。从同一个 `SdkConfig` 创建的客户端共享一个 `IdentityCache`。

在中使用行为版本 适用于 Rust 的 AWS SDK

适用于 Rust 的 AWS SDK 开发人员期望并依赖该语言及其主要库提供的强大且可预测的行为。为了帮助使用适用于 Rust 的 SDK 的开发人员获得预期行为，客户端配置需要包含 `BehaviorVersion`。`BehaviorVersion` 指定预期为默认值的 SDK 版本。这使得 SDK 可以不断发展，改变最佳实践以匹配新标准并支持新功能，而不会对应用程序的行为产生意想不到的不利影响。

> **Warning**
>
> 如果您尝试配置 SDK 或创建客户端，但未明确指定 `BehaviorVersion`，则构造函数会 panic。

例如，假设发布了新版本的 SDK，并采用了新的默认重试策略。如果您的应用程序使用与先前版本的 SDK 匹配的 `BehaviorVersion`，则将使用该先前配置而不是新的默认配置。

每次发布适用于 Rust 的 SDK 的新行为版本时，之前的 BehaviorVersion 都会被标记为适用于 Rust 的 SDK deprecated 属性，并且会添加新版本。这会导致在编译时出现警告，但除此之外，构建过程会照常进行。BehaviorVersion::latest() 也会更新，以指示新版本的默认行为。

> **Note**
>
> 如果您的代码不依赖于极其具体的行为特性，则应在代码中使用 BehaviorVersion::latest()，或者在 Cargo.toml 文件中使用功能标志 behavior-version-latest。如果您要编写对延迟敏感的代码或用于调整 Rust SDK 行为的代码，可以考虑将 BehaviorVersion 固定到特定的主要版本。

在 Cargo.toml 中设置行为版本

您可以通过在 Cargo.toml 文件中添加相应的功能标志，为 SDK 和各个模块（例如 aws-sdk-s3 或 aws-sdk-iam）指定行为版本。目前，Cargo.toml 仅支持 latest 版本的 SDK。

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

在代码中设置行为版本

您可以通过在配置 SDK 或客户端时指定行为版本，根据需要更改代码的行为版本：

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

此示例创建了一个配置，该配置使用环境来配置 SDK，但将 BehaviorVersion 设置为 v2023_11_09()。

在 Rust AWS 开发工具包中配置重试次数

适用于 Rust 的 AWS SDK 为服务请求提供了默认的重试行为和可自定义的配置选项。调用 AWS 服务偶尔会返回意外异常。如果重试调用，某些类型的错误（例如节流或暂时错误）可能会成功。

重试行为可以使用共享 AWS config文件中的环境变量或设置进行全局配置。有关此方法的信息，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的重试行为。它还包括有关重试策略实现以及如何选择一种策略而不是另一种策略的详细信息。

或者，您也可以在代码中配置这些选项，如以下部分所示。

默认重试配置

每个服务客户端都默认使用通过 RetryConfig 结构提供的 standard 重试策略配置。默认情况下，一个调用将尝试三次（初次尝试加上两次重试）。此外，为了避免重试风暴，每次重试都将随机延迟一小段时间。此约定适用于大多数使用案例，但可能不适用于特定情况，例如高吞吐量系统。

只有某些类型的错误被认为是可重试的。 SDKs可重试错误示例如下：

- 套接字超时
- 服务端节流
- 暂时性服务错误，例如 HTTP 5XX 响应

以下示例不属于可重试范畴：

- 参数缺失或无效
- 身份验证/安全错误
- 配置错误异常

您可以通过设置最大尝试次数、延迟和回退配置来自定义 standard 重试策略。

最大尝试次数

您可以通过向 aws_config::defaults 提供修改后的 RetryConfig 来自定义代码中的最大尝试次数：

```
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;
```

延迟和回退

如果需要重试，默认的重试策略会在进行下一次尝试之前等待一段时间。第一次重试的延迟时间很短，但后续重试的延迟时间会呈指数级增长。最长延迟时间是有上限的，以防延迟时间过长。

系统会将随机抖动应用于所有尝试之间的延迟。抖动有助于降低可能导致重试风暴的大量重试的影响。有关指数回退和抖动的更深入讨论，请参阅 AWS 架构博客中的指数回退和抖动。

您可以通过向 `aws_config::defaults` 提供修改后的 `RetryConfig` 来自定义代码中的延迟设置。以下代码将延迟配置设置为第一次重试尝试最多延迟 100 毫秒，并且任何重试尝试之间的最大时间间隔为 5 秒。

```
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;
```

自适应重试模式

作为 `standard` 模式重试策略的替代方案，`adaptive` 模式重试策略是一种高级方法，旨在寻找理想的请求速率以尽可能减少节流错误。

> **Note**
>
> 自适应重试是一种高级重试模式。通常不建议使用此策略。请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的重试行为。

自适应重试具备标准重试的所有功能。它添加了一个客户端速率限制器，用于衡量节流请求与非节流请求的比率。它还会限制尝试流量以保持在安全带宽内，理想情况下会使节流错误为零。

该速率会根据不断变化的服务条件和流量模式实时调整，并可能相应地增加或减少流量速率。至关重要的是，在高流量场景中，速率限制器可能会延迟初始尝试。

您可以通过提供修改后的 `RetryConfig` 来选择代码中的 `adaptive` 重试策略：

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

在 Rust AWS 开发工具包中配置超时

适用于 Rust 的 AWS SDK 提供了多种用于管理 AWS 服务 请求超时和停滞数据流的设置。当网络中发生意外延迟和故障时，这些设置可以帮助应用程序以最佳状态运行。

API 超时

当存在可能导致请求尝试花费很长时间或完全失败的暂时性问题时，请务必检查和设置超时，这样应用程序才能实施快速失效机制并以最佳状态运行。SDK 可以自动重试失败的请求。为单个尝试和整个请求都设置超时是一种很好的做法。

适用于 Rust 的 SDK 提供了为请求建立连接的默认超时。SDK 没有为接收请求尝试或整个请求的响应设置任何默认最大等待时间。以下超时选项可用：

参数	默认 值	说明
连接超时	3.1 秒	放弃连接前等待建立连接的最长时间。
操作超时	无	接收来自适用于 Rust 的 SDK 的响应之前要等待的最长时间，包括所有重试。
操作尝试超时	无	等待单次 HTTP 尝试的最长时间，超过该时长之后可以重试 API 调用。
读取超时	无	自请求发起到读取响应的第一个字节的最长时间。

以下示例演示了使用自定义超时值配置 Amazon S3 客户端的过程。

```
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);
```

将这些操作和尝试超时一起使用时，可以对所有重试尝试所花费的总时间设置硬性限制。您还可以将单个 HTTP 请求设置为在慢速请求时快速失败。

除了在服务客户端上为所有操作设置这些超时值之外，您还可以为单个请求配置或覆盖这些超时值。

> **Important**
>
> 对于在适用于 Rust 的 SDK 返回响应后使用的流数据，操作和尝试超时不适用。例如，使用来自响应 `ByteStream` 成员的数据不受操作超时的影响。

停滞流保护

适用于 Rust 的 SDK 提供了另一种与检测停滞流相关的超时机制。停滞流是指在超过配置的宽限期后仍未生成任何数据的上传或下载流。这有助于防止应用程序无限期挂起而永远无法取得进展。

停滞流保护会在流空闲时间超过可接受时段时返回错误。

默认情况下，适用于 Rust 的 SDK 为上传和下载启用停滞的直播保护，并在 20 秒 byte/sec 的宽限期内查找至少 1 个活动。

以下示例展示了禁用上传保护并将无活动宽限期更改为 10 秒的自定义 `StalledStreamProtectionConfig`：

```
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**
>
> 停滞流保护是一个高级配置选项。建议仅在应用程序对性能要求更高或这些值会导致一些其他问题时才更改这些值。

禁用停滞流保护

以下示例展示了如何完全禁用停滞流保护：

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

在适用于 Rust 的 AWS SDK 中配置可观测性功能

可观测性是指可以根据系统发出的数据推断出其当前状态的程度。发出的数据通常称为遥测。

主题

- 在适用于 Rust 的 AWS SDK 中配置和使用日志记录

在适用于 Rust 的 AWS SDK 中配置和使用日志记录

适用于 Rust 的 AWS SDK 使用跟踪框架进行日志记录。要启用日志记录，请添加 `tracing-subscriber` crate 并在 Rust 应用程序中对其进行初始化。日志包括时间戳、日志级别和模块路径，有助于调试 API 请求和 SDK 行为。使用 `RUST_LOG` 环境变量控制日志的详细程度，必要时按模块筛选。

如何在适用于 Rust 的 AWS SDK 中启用日志记录

1. 在项目目录的命令提示符中，将 tracing-subscriber crate 添加为依赖项：

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

   这会将 crate 添加到 `Cargo.toml` 文件的 `[dependencies]` 部分。

2. 初始化订阅用户。通常，这是在调用任何适用于 Rust 的 SDK 操作之前在 `main` 函数中提早完成的：

```
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. 使用 RUST_LOG 环境变量启用日志记录。要启用日志记录信息的显示，请在命令提示符中将 RUST_LOG 环境变量设置为要记录的级别。以下示例将日志记录设置为 debug 级别。

   Linux/macOS

```
$ RUST_LOG=debug
```

   Windows

   如果您使用的是 VSCode，则终端窗口通常默认为 PowerShell。验证您使用的提示类型。

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

   PowerShell

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

4. 运行程序：

```
$ cargo run
```

您应该在控制台或终端窗口中看到其他输出。

有关更多信息，请参阅 tracing-subscriber 文档中的使用环境变量筛选事件。

解读日志输出

按照上一节中的步骤启用日志记录后，默认情况下，其他日志信息将打印为标准输出。

如果您使用的是默认日志输出格式（跟踪模块称之为“完整”），则您在日志输出中看到的信息类似于以下内容：

```
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
```

每个条目都包含以下值：

- 日志条目的时间戳。
- 条目的日志级别。这是一个单词，例如 INFO、DEBUG 或 TRACE。
- 生成日志条目的嵌套跨度集，用冒号(“:”)分隔。这可帮助您确定日志条目的来源。
- 包含生成日志条目的 Rust 模块路径。
- 日志消息文本。

跟踪模块的标准输出格式使用 ANSI 转义码对输出进行着色。筛选或搜索输出时，请记住这些转义序列。

> **Note**
>
> 嵌套跨度集中显示的 sdk_invocation_id 是由 SDK 在客户端生成的唯一 ID，用于帮助关联日志消息。它与在 AWS 服务的响应中找到的请求 ID 无关。

微调日志记录级别

如果您使用支持环境筛选的 crate（例如 `tracing_subscriber`），则可以按模块控制日志的详细程度。

您可以为每个模块开启相同的日志记录级别。以下内容将每个模块的日志记录级别设置为了 `trace`：

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

您可以为特定模块开启跟踪级别日志记录。在以下示例中，只有来自 `aws_smithy_runtime` 的日志才会进入 `trace` 级别。

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

您可以用逗号分隔多个模块，从而为它们指定不同的日志级别。以下示例将 `aws_config` 模块设置为 `trace` 级别日志记录，将 `aws_smithy_runtime` 模块设置为 `debug` 级别日志记录。

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

下表列出了可用于筛选日志消息的一些模块：

Prefix	描述
`aws_smithy_runtime`	请求和响应线路日志记录
`aws_config`	凭证加载
`aws_sigv4`	请求签名和规范请求

要确定需要在日志输出中包含哪些模块，一种方法是先记录所有内容，然后在日志输出中找到 crate 名称以获取所需的信息。然后，您可以相应地设置环境变量并再次运行程序。

在中配置客户端终端节点 适用于 Rust 的 AWS SDK

当 适用于 Rust 的 AWS SDK 调用 a 时 AWS 服务，其第一步之一就是确定将请求路由到何处。此过程称为端点解析。

在创建服务客户端时，您可以为 SDK 配置端点解析。端点解析的默认配置通常没问题，但您可能出于多种原因需要修改默认配置。两个示例原因如下所示：

- 向服务的预发行版本或服务的本地部署发出请求。
- 访问尚未在 SDK 中建模的特定服务功能。

> **Warning**
>
> 端点解析是一个高级 SDK 主题。如果您更改默认设置，则可能会破坏代码。默认设置应用于生产环境中的大多数用户。

可以全局设置自定义终端节点，以便它们用于所有服务请求，也可以为特定的终端节点设置自定义终端节点 AWS 服务。

可以使用共享 AWS `config`文件中的环境变量或设置来配置自定义终端节点。有关此方法的信息，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的服务特定端点。有关所有共享`config`文件设置和环境变量的完整列表 AWS 服务，请参阅服务特定端点的标识符。

或者，您也可以在代码中配置此自定义选项，如以下部分所示。

自定义配置

您可以使用在构建客户端时可用的两种方法来自定义服务客户端的端点解析：

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

您可以设置这两个属性。但是，大多数情况下，您只提供一个。对于常规使用情况，endpoint_url 通常需要自定义。

设置端点 URL

您可以为 endpoint_url 设置一个值来表示服务的“基本”主机名。但是，此值不是最终值，因为它是作为参数传递给客户端 ResolveEndpoint 实例的。然后，ResolveEndpoint 实现会检查该值，并可能修改该值以确定最终端点。

设置端点解析器

服务客户端的 ResolveEndpoint 实现决定了 SDK 用于任何给定请求的最终解析端点。服务客户端为每个请求调用 resolve_endpoint 方法，并使用解析器返回的 EndpointFuture 值，而不做任何更改。

以下示例演示了如何为 Amazon S3 客户端提供自定义端点解析器实现，该实现会根据不同的阶段（例如暂存和生产）解析不同的端点：

```
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**
>
> 端点解析器以及由此扩展的 `ResolveEndpoint` 特性是每个服务特有的，因此只能在服务客户端配置中进行配置。另一方面，可以使用共享配置（应用于从中派生的所有服务）或针对特定服务配置端点 URL。

ResolveEndpoint 参数

`resolve_endpoint` 方法接受包含端点解析中使用的属性的服务特定参数。

每个服务均包含以下基本属性：

Name	Type	说明
`region`	字符串	客户的 AWS 区域
`endpoint`	字符串	`endpointUrl` 的值集的字符串表示形式
`use_fips`	布尔值	是否在客户端配置中启用了 FIPS 端点
`use_dual_stack`	布尔值	是否在客户端配置中启用了双栈端点

AWS 服务 可以指定解析所需的其他属性。例如，Amazon S3 端点参数包括存储桶名称和几个特定于 Amazon S3 的功能设置。例如，`force_path_style` 属性决定是否可以使用虚拟主机寻址。

如果您实施自己的提供程序，则无需构建自己的端点参数实例。SDK 为每个请求提供属性并将其传递给您的 `resolve_endpoint` 实现。

将使用 `endpoint_url` 与使用 `endpoint_resolver` 进行比较

需要注意的是，以下两种配置（一种使用 `endpoint_url`，另一种使用 `endpoint_resolver`）产生的客户端在端点解析行为上并不相同。

```
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();
```

设置 `endpoint_url` 的客户端会指定传递给（默认）提供程序的基本 URL，该 URL 可以作为端点解析的一部分进行修改。

设置 `endpoint_resolver` 的客户端指定 Amazon S3 客户端使用的最终 URL。

示例

自定义端点通常用于测试。不是向基于云的服务发送调用，而是将调用路由到本地托管的模拟服务。其中两种选项是：

- DynamoDB local – Amazon DynamoDB 服务的本地版本。
- LocalStack— 在本地计算机的容器中运行的云服务模拟器。

以下示例说明了指定自定义端点以使用这两个测试选项的两种不同方法。

直接在代码中使用 DynamoDB local

如前几节所述，您可以直接在代码中设置 `endpoint_url` 来覆盖基本端点，以便指向本地 DynamoDB 服务器。在代码中：

```
    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);
```

完整的示例可在上找到 GitHub。

LocalStack 使用该**config**文件

您可以在共享 AWS config文件中设置特定于服务的终端节点。以下配置的配置文件将 endpoint_url 设置为连接到端口 4566 上的 localhost。有关 LocalStack 配置的更多信息，请参阅LocalStack文档网站上的 LocalStack 通过端点 URL 进行访问。

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

当您使用 localstack 配置文件时，SDK 将获取共享 config 文件中的更改并将其应用到您的 SDK 客户端。使用这种方法，您的代码无需包含对端点的任何引用，如下所示：

```
    // 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);
```

完整的示例可在上找到 GitHub。

在适用于 Rust 的 AWS SDK 中覆盖客户端的单个操作配置

创建服务客户端后，配置将变为不可变并将应用于所有后续操作。虽然此时无法修改配置，但可以按操作对其进行覆盖。

每个操作生成器都有一种可用于创建 CustomizableOperation 的 customize 方法，以便您可以覆盖现有配置的单个副本。原始客户端配置将保持不变。

以下示例展示了创建 Amazon S3 客户端的过程，该客户端调用两个操作，其中第二个操作被覆盖以发送到另一个 AWS 区域。Amazon S3 的所有对象调用都使用 us-east-1 区域，除非 API 调用被显式覆盖以使用修改后的 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**
>
> 上述示例针对的是 Amazon S3，但是概念对所有操作都是一样的。某些操作可能在 CustomizeableOperation 方面有其他方法。

有关使用 customize 为单个操作添加拦截器的示例，请参阅仅用于拦截特定操作的拦截器。

在 Rust AWS 开发工具包中配置 HTTP 级别的设置

适用于 Rust 的 AWS SDK 提供了内置 HTTP 功能，供您在代码中创建的 AWS 服务 客户端使用。

默认情况下，适用于 Rust 的 SDK 使用基于 hyper、rustls 和 aws-lc-rs 的 HTTPS 客户端。此客户端应该适用于大多数使用案例，而无需其他配置。

- hyper是 Rust 的较低级别 HTTP 库，可以与一起使用 适用于 Rust 的 AWS SDK 来调用 API 服务。
- rustls 是一个用 Rust 编写的现代 TLS 库，它为加密提供程序提供了内置选项。
- aws-lc 是一个通用加密库，包含 TLS 和常见应用程序所需的算法。
- aws-lc-rs 是 Rust 中用于 aws-lc 库的惯用包装器。

如果您想选择其他 TLS 或加密提供程序，aws-smithy-http-client 提供了一些其他选项和配置。对于更高级的使用案例，我们鼓励您自带 HTTP 客户端实现或提交功能申请以供考虑。

选择替代 TLS 提供程序

aws-smithy-http-client crate 提供了一些替代 TLS 提供程序。

提供以下提供程序：

rustls 与 aws-lc

基于 rustls 的 TLS 提供程序，使用 aws-lc-rs 进行加密。

这是适用于 Rust 的 SDK 的默认 HTTP 行为。如果您想使用此选项，则无需在代码中执行任何其他操作。

s2n-tls

基于 s2n-tls 的 TLS 提供程序。

rustls 与 aws-lc-fips

基于 rustls 的 TLS 提供程序，使用符合 FIPS 标准的 aws-lc-rs 版本进行加密

rustls 与 ring

基于 rustls 的 TLS 提供程序，使用 ring 进行加密。

先决条件

使用 `aws-lc-rs` 或 `s2n-tls` 需要 C 编译器（Clang 或 GCC）来构建。对于某些平台，构建可能还需要 CMake。在任何平台上使用 “fips” 功能进行构建都需要 CMake 然后开始。有关更多信息，请参阅适用于 Rust 的 AWS Libcrypto（`aws-lc-rs`）存储库和构建说明。

如何使用替代 TLS 提供程序

`aws-smithy-http-client` crate 提供了其他 TLS 选项。要让您的 AWS 服务 客户端使用其他 TLS 提供程序，请使用 `aws_config` crate 中的加载器覆盖 `http_client`。HTTP 客户端既用于 AWS 服务 ，也用于凭证提供程序。

以下示例演示了如何使用 `s2n-tls` TLS 提供程序。但是，类似的方法也适用于其他提供程序。

要编译示例代码，请运行以下命令向项目添加依赖项：

```
cargo add aws-smithy-http-client -F s2n-tls
```

示例代码：

```
use aws_smithy_http_client::{tls, Builder};

#[tokio::main]
async fn main() {
    let http_client = Builder::new()
        .tls_provider(tls::Provider::S2nTls)
        .build_https();

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

    // create client(s) using sdk_config
    // e.g. aws_sdk_s3::Client::new(&sdk_config);
}
```

启用 FIPS 支持

`aws-smithy-http-client` crate 提供了启用符合 FIPS 标准的加密实现的选项。要让您的 AWS 服务 客户使用符合 FIPS 标准的提供商，请`http_client`使用箱子中的加载器替换。`aws_config`HTTP 客户端既 AWS 服务 用于证书提供者，也用于凭证提供程序。

> **Note**
>
> FIPS 支持需要在构建环境中添加其他依赖项。请参阅 `aws-lc` crate 的构建说明。

要编译示例代码，请运行以下命令向项目添加依赖项：

```
cargo add aws-smithy-http-client -F rustls-aws-lc-fips
```

以下示例代码可启用 FIPS 支持：

```
// file: main.rs
use aws_smithy_http_client::{
    tls::{self, rustls_provider::CryptoMode},
    Builder,
};

#[tokio::main]
async fn main() {
    let http_client = Builder::new()
        .tls_provider(tls::Provider::Rustls(CryptoMode::AwsLcFips))
        .build_https();

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

    // create client(s) using sdk_config
    // e.g. aws_sdk_s3::Client::new(&sdk_config);
}
```

优先考虑后量子密钥交换

默认 TLS 提供程序基于使用 `aws-lc-rs` 的 `rustls`，支持 X25519MLKEM768 后量子密钥交换算法。要使 X25519MLKEM768 成为优先级最高的算法，您需要将 `rustls` 程序包添加到 crate 中并启用 `prefer-post-quantum` 功能标志。否则，它可用，但不是最高优先级。有关更多信息，请参阅`rustls`文档。

> **Note**
>
> 这将在未来版本中成为默认选项。

覆盖 DNS 解析器

通过手动配置 HTTP 客户端，可以覆盖默认 DNS 解析器。

要编译示例代码，请运行以下命令向项目添加依赖项：

```
cargo add aws-smithy-http-client -F rustls-aws-lc
cargo add aws-smithy-runtime-api -F client
```

以下示例代码覆盖 DNS 解析器：

```
use aws_smithy_http_client::{
    tls::{self, rustls_provider::CryptoMode},
    Builder
};
use aws_smithy_runtime_api::client::dns::{DnsFuture, ResolveDns};
use std::net::{IpAddr, Ipv4Addr};

/// A DNS resolver that returns a static IP address (127.0.0.1)
#[derive(Debug, Clone)]
struct StaticResolver;

impl ResolveDns for StaticResolver {
    fn resolve_dns<'a>(&'a self, _name: &'a str) -> DnsFuture<'a> {
        DnsFuture::ready(Ok(vec![IpAddr::V4(Ipv4Addr::new(127, 0, 0, 1))]))
    }
}

#[tokio::main]
```

```
async fn main() {
    let http_client = Builder::new()
        .tls_provider(tls::Provider::Rustls(CryptoMode::AwsLc))
        .build_with_resolver(StaticResolver);

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

    // create client(s) using sdk_config
    // e.g. aws_sdk_s3::Client::new(&sdk_config);
}
```

> **Note**
>
> 默认情况下，亚马逊 Linux 2023 (AL2023) 不会在操作系统级别缓存 DNS。

自定义根 CA 证书

默认情况下，TLS 提供程序会加载给定平台的系统原生根证书。要自定义此行为以加载自定义 CA 捆绑包，您可以使用自己的 `TrustStore` 配置 `TlsContext`。

要编译示例代码，请运行以下命令向项目添加依赖项：

```
cargo add aws-smithy-http-client -F rustls-aws-lc
```

以下示例结合使用了 `rustls` 和 `aws-lc`，但也适用于任何受支持的 TLS 提供程序：

```
use aws_smithy_http_client::{
    tls::{self, rustls_provider::CryptoMode},
    Builder
};
use std::fs;

/// read the PEM encoded root CA (bundle) and return a custom TLS context
fn tls_context_from_pem(filename: &str) -> tls::TlsContext {
    let pem_contents = fs::read(filename).unwrap();
```

```
    // Create a new empty trust store (this will not load platform native certificates)
    let trust_store = tls::TrustStore::empty()
        .with_pem_certificate(pem_contents.as_slice());

    tls::TlsContext::builder()
        .with_trust_store(trust_store)
        .build()
        .expect("valid TLS config")
}

#[tokio::main]
async fn main() {
    let http_client = Builder::new()
        .tls_provider(tls::Provider::Rustls(CryptoMode::AwsLc))
        .tls_context(tls_context_from_pem("my-custom-ca.pem"))
        .build_https();

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

    // create client(s) using sdk_config
    // e.g. aws_sdk_s3::Client::new(&sdk_config);
}
```

在 Rust AWS 开发工具包中配置拦截器

您可以使用拦截器介入 API 请求和响应的执行过程。拦截器是一种开放式机制，在这种机制中，SDK 调用您编写的代码将行为注入 request/response 生命周期。这样，您便可修改正在进行的请求、调试请求处理、查看错误等。

以下示例显示了一个简单的拦截器，该拦截器会在进入重试循环之前向所有传出请求添加一个额外的标头：

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

有关更多信息和可用的拦截器钩子，请参阅拦截特性。

拦截器注册

构造服务客户端或覆盖特定操作的配置时，可以注册拦截器。注册方式会有所不同，具体取决于您希望拦截器应用于客户端的所有操作，还是仅应用于特定操作。

用于拦截服务客户端上所有操作的拦截器

要为整个客户端注册拦截器，请使用 Builder 模式添加拦截器。

```
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);
```

仅用于拦截特定操作的拦截器

要仅为单个操作注册拦截器，请使用 customize 扩展程序。您可以使用此方法在每个操作级别覆盖服务客户端配置。有关可自定义的操作的更多信息，请参阅在适用于 Rust 的 AWS SDK 中覆盖客户端的单个操作配置。

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

使用适用于 Rust 的 AWS SDK

了解将适用于 Rust 的 AWS SDK 与 AWS 服务配合使用的常见方法和推荐方法。

主题

- 使用适用于 Rust 的 AWS SDK 发出 AWS 服务请求
- 使用适用于 Rust 的 AWS SDK 的最佳实践
- 中的并发性 适用于 Rust 的 AWS SDK
- 在适用于 Rust 的 AWS SDK 中创建 Lambda 函数
- 使用适用于 Rust 的 AWS SDK 创建预签名 URL
- 处理 Rust AWS 开发工具包中的错误
- 在适用于 Rust 的 AWS SDK 中使用分页结果
- 向适用于 Rust 的 AWS SDK 应用程序添加单元测试
- 在适用于 Rust 的 AWS SDK 中使用等待器

使用适用于 Rust 的 AWS SDK 发出 AWS 服务请求

要以编程方式访问 AWS 服务，适用于 Rust 的 AWS SDK 对每个 AWS 服务使用一个客户端结构体。例如，如果您的应用程序需要访问 Amazon EC2，则您的应用程序会创建一个 Amazon EC2 客户端结构体来与该服务交互。然后，您可以使用服务客户端向该 AWS 服务 发出请求。

要向 AWS 服务发出请求，您必须先创建和配置服务客户端。对于您的代码使用的每个 AWS 服务，它都有自己的 crate 和用于与之交互的专用类型。客户端为服务公开的每个 API 操作公开一种方法。

要在适用于 Rust 的 AWS SDK 中与 AWS 服务交互，请创建一个特定于服务的客户端，使用其 API 方法和流畅的生成器式串接，并调用 `send()` 以执行请求。

`Client` 为服务公开的每个 API 操作公开一种方法。这些方法的返回值都是“fluent builder”，通过生成器式函数调用串接为该 API 添加不同的输入。调用服务的方法后，调用 `send()` 以获取 `Future`，它将产生成功的输出或 `SdkError`。有关 `SdkError` 的更多信息，请参阅 处理 Rust AWS 开发工具包中的错误。

以下示例展示了使用 Amazon S3 在 `us-west-2` AWS 区域创建存储桶的基本操作：

```
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;
```

每个服务 crate 都有其他用于 API 输入的模块，例如：

- `types` 模块具有结构体或枚举来提供更复杂的结构化信息。
- `primitives` 模块具有更简单的类型来表示数据，例如日期时间或二进制 blob。

有关服务 crate 及 crate 组织的更详细信息，请参阅 API 参考文档。例如，Amazon Simple Storage Service 的 `aws-sdk-s3` crate 具有多个模块。其中两个是：

- `aws_sdk_s3::types`
- `aws_sdk_s3::primitives`

使用适用于 Rust 的 AWS SDK 的最佳实践

以下是使用适用于 Rust 的 AWS SDK 的最佳实践。

尽可能重复使用 SDK 客户端

根据 SDK 客户端的构造方式，创建新客户端可能会导致每个客户端维护自己的 HTTP 连接池、身份缓存等。我们建议共享客户端或至少共享 `SdkConfig`，以避免创建昂贵资源所带来的开销。所有 SDK 客户端都将 `Clone` 实现为单个原子引用计数更新。

配置 API 超时

SDK 为某些超时选项（例如连接超时和套接字超时）提供默认值，但不为 API 调用超时或单个 API 调用尝试提供默认值。为单个尝试和整个请求都设置超时是一种很好的做法。当存在可能导致请求尝试花费更长时间才能完成的临时问题或出现严重的网络问题时，这将确保您的应用程序以最佳方式快速失败。

有关配置操作超时的更多信息，请参阅在 Rust AWS 开发工具包中配置超时。

中的并发性 适用于 Rust 的 AWS SDK

适用于 Rust 的 AWS SDK 不提供并发控制，但用户有很多选择可以实现自己的并发控制。

术语

与该主题相关的术语很容易混淆，有些术语虽然最初代表不同的概念，但现在已经变成了同义词。在本指南中，我们将定义以下内容：

- 任务：您的程序将运行至完成或尝试运行至完成的某些“工作单元”。
- 顺序计算：一个接一个地执行多个任务。
- 并行计算：在重叠的时间段内执行多个任务。
- 并发性：计算机以任意顺序完成多个任务的能力。
- 多任务处理：计算机同时运行多个任务的能力。
- 竞态条件：程序的行为根据任务启动时间或处理任务所需的时间而发生变化。
- 争用：因访问共享资源而发生的冲突。当两个或多个任务想要同时访问一个资源时，该资源即处于“争用状态”。
- 死锁：一种无法取得更多进展的状态。之所以发生这种情况，通常是因为两个任务都想要获取对方的资源，但在另一个任务的资源可用之前，两个任务都不会释放自己的资源。死锁会导致程序部分或完全没有响应。

一个简单示例

我们的第一个示例是一个顺序程序。在后面的示例中，我们将使用并发技术更改此代码。后面的示例重复使用相同的 `build_client_and_list_objects_to_download()` 方法并在 `main()` 中进行更改。运行以下命令来添加项目的依赖项：

- `cargo add aws-sdk-s3`

- `cargo add aws-config tokio --features tokio/full`

以下示例任务是下载 Amazon Simple Storage Service 存储桶中的所有文件：

1. 首先列出所有文件。将密钥保存在列表中。
2. 遍历列表，依次下载每个文件

```
use aws_sdk_s3::{Client, Error};
const EXAMPLE_BUCKET: &str = "amzn-s3-demo-bucket";  // Update to name of bucket you
 own.

// 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() -> (Client, Vec<String>) {
    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**
>
> 在这些示例中，我们不会处理错误，并且我们假设示例存储桶中没有密钥看起来像文件路径的对象。因此，我们不会介绍如何创建嵌套目录。

由于现代化计算机的架构，我们可以重写这个程序以提高效率。我们将在后面的示例中这样做，但首先，让我们再学习几个概念。

所有权和可变性

Rust 中的每个值都有一个所有者。当所有者超出范围时，其拥有的所有值也将被删除。所有者可以提供一个或多个指向某个值的不可变引用，或者提供一个可变引用。Rust 编译器负责确保任何引用的有效期都不会超过其所有者。

当多个任务需要以可变方式访问同一个资源时，需要进行额外的规划和设计。在顺序计算中，每个任务都能以可变方式访问相同的资源而不会发生争用，因为它们按顺序一个接一个地运行。但是，在并行计算中，任务可以按任意顺序同时运行。因此，我们必须做更多的工作来向编译器证明多个可变引用是不可能的（或者在发生这种情况时至少会导致崩溃）。

Rust 标准库提供了许多工具来帮助我们实现这一目标。有关这些主题的更多信息，请参阅《Rust 编程语言》一书中的变量与可变性以及了解所有权。

更多术语！

以下是“同步对象”列表。总而言之，它们是使编译器相信我们的并发程序不会违反所有权规则所必需的工具。

标准库同步对象：

- Arc：一个原子引用计数指针。当数据封装在 Arc 中时，可以自由共享，而不必担心任何特定的所有者会提早删除该值。从这个意义上讲，值的所有权变得“共享”。Arc 中的值不能是可变的，但可能具有内部可变性。
- 屏障：确保多个线程相互等待对方到达程序中的某个点，然后再继续一起执行。
- 条件变量：一个条件变量，支持在等待事件发生时屏蔽线程。
- 互斥锁：一种双向排除机制，可确保一次最多一个线程能够访问某些数据。一般而言，不要将 Mutex 锁放在代码中的某个 .await 点上。

Tokio 同步对象：

虽然 AWS SDKs 旨在与async运行时无关，但我们建议在特定情况下使用tokio同步对象。

- 互斥锁：与标准库的 Mutex 类似，但成本稍高。与标准 Mutex 不同，这个可以放在代码中的某个 .await 点上。
- 信号量：一个用于控制多个任务对公共资源的访问的变量。

重写我们的示例以提高效率（单线程并发）

在以下修改后的示例中，我们使用 futures_util::future::join_all 并发运行所有get_object请求。运行以下命令来添加项目的新依赖项。

- cargo add futures-util

```
#[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;
}
```

这是从并发中受益的最简单方法，但它也有一些最初可能并不明显的问题：

1. 我们同时创建所有请求输入。如果我们没有足够的内存来容纳所有get_object请求输入，那么我们将遇到 “out-of-memory” 分配错误。
2. 我们同时创造和等待所有 Future。如果我们试图一次下载过多内容，Amazon S3 会限制请求。

要解决这两个问题，我们必须限制同时发送的请求数量。我们将使用 tokio 信号量来做到这一点：

```
use std::sync::Arc;
use tokio::sync::Semaphore;
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;
}
```

我们通过将请求创建移到 async 数据块中，解决了潜在内存使用问题。这样，只有在需要发送请求时才会创建请求。

> **Note**
>
> 如果内存足够，一次性创建所有请求输入并将它们保存在内存中，直到准备好发送为止，可能会更有效率。要尝试此操作，请将请求输入创建移到 async 数据块之外。

我们还通过将传输中的请求限制为发送至 CONCURRENCY_LIMIT，解决了同时发送过多请求的问题。

> **Note**
>
> 每个项目的正确 CONCURRENCY_LIMIT 值都不一样。在构建和发送自己的请求时，尽量将其设置得尽可能高，以免遇到节流错误。虽然可以根据服务返回的成功响应与节流响应的比率动态更新并发限制，但由于其复杂性，这超出了本指南的范围。

重写我们的示例以提高效率（多线程并发）

在前两个示例中，我们并行执行了请求。虽然这比同步运行效率更高，但我们可以使用多线程来进一步提高效率。要使用 tokio 实现此目标，我们需要将它们作为单独的任务生成。

> **Note**
>
> 此示例要求您使用多线程 tokio 运行时。此运行时需要启用 rt-multi-thread 功能才能使用。当然，您需要在多核机器上运行您的程序。

运行以下命令来添加项目的新依赖项。

- `cargo add tokio --features=rt-multi-thread`

```
// 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;
}
```

将工作划分为任务可能很复杂。执行 I/O （输入/输出）通常是阻塞的。运行时可能很难在长时间运行的任务的需求和短期任务的需求之间取得平衡。无论您选择哪种运行时，请务必阅读其建议，以最有效的方式将您的工作划分为任务。有关 tokio 运行时建议，请参阅模块 tokio::task。

调试多线程应用程序

并发运行的任务可以按任意顺序运行。因此，并发程序的日志可能很难读取。在适用于 Rust 的 SDK 中，我们建议使用 tracing 日志记录系统。它可以根据特定任务将日志分组，而无论它们何时运行。有关指南，请参阅在适用于 Rust 的 AWS SDK 中配置和使用日志记录。

识别已锁定任务的一个非常有用的工具是 tokio-console，它是异步 Rust 程序的诊断和调试工具。通过检测和运行程序，然后运行 tokio-console 应用程序，您可以看到程序正在运行的任务的实时视图。此视图包含有用的信息，例如任务等待获取共享资源所花费的时间或被轮询的次数。

在适用于 Rust 的 AWS SDK 中创建 Lambda 函数

有关使用适用于 Rust 的 AWS SDK 开发 AWS Lambda 函数的详细文档，请参阅《 AWS Lambda 开发人员指南》中的使用 Rust 构建 Lambda 函数。该文档将指导您：

- 使用 Rust Lambda 运行时客户端 crate 获取核心功能 aws-lambda-rust-runtime。
- 使用推荐的命令行工具，借助 Cargo Lambda 将 Rust 函数二进制文件部署到 Lambda。

除了《AWS Lambda 开发人员指南》中的指导示例外，GitHub 上的 AWS SDK 代码示例存储库中还提供了 Lambda 计算器示例。

使用适用于 Rust 的 AWS SDK 创建预签名 URL

您可以对某些 AWS API 操作的请求进行预签名，以便其他调用方以后无需提供自己的凭证即可使用该请求。

例如，假设 Jane 有权访问 Amazon Simple Storage Service（Amazon S3）对象，并希望临时与 Alejandro 分享对该对象的访问权限。Jane 可以生成预签名的 `GetObject` 请求来分享给 Alejandro，这样 Alejandro 就可以下载该对象而无需访问 Jane 的凭证，也无需拥有自己的凭证。预签名 URL 使用的凭证是 Jane 的，因为她是生成该 URL 的 AWS 用户。

要了解有关 Amazon S3 中预签名 URL 的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用预签名 URL。

预签名基础知识

适用于 Rust 的 AWS SDK 在操作 fluent-builders 上提供了一个 `presigned()` 方法，可用于获取预签名请求。

以下示例展示了为 Amazon S3 创建预签名 `GetObject` 请求的过程。请求在创建后的 5 分钟内有效。

```
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()` 方法返回 `Result<PresignedRequest, SdkError<E, R>>`。

返回的 `PresignedRequest` 包含用于获取 HTTP 请求组件的方法，包括方法、URI 和任何标头。所有这些都需要发送到相应服务（如果有），请求才会生效。不过，许多预签名请求可以单独由 URI 表示。

预签名 POST 和 PUT 请求

许多可预签名的操作只需要一个 URL，并且必须作为 HTTP GET 请求发送。但是，有些操作需要正文，在某些情况下必须作为 HTTP POST 或 HTTP PUT 请求与标头一起发送。对这些请求进行预签名与对 GET 请求进行预签名相同，但是调用预签名请求更为复杂。

以下示例展示了对 Amazon S3 PutObject 请求进行预签名并将其转换为可使用所选 HTTP 客户端发送的 http::request::Request 请求的过程。

要使用 into_http_1x_request() 方法，请将 http-1x 功能添加到 Cargo.toml 文件中的 aws-sdk-s3 crate。

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

源文件：

```
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);
```

独立签署人

> **Note**
>
> 这是一个高级使用案例。对于大多数用户来说，既不需要也不建议使用。

在一些使用案例中，需要在适用于 Rust 的 SDK 上下文之外创建签名请求。为此，您可以独立于 SDK 使用 aws-sigv4 crate。

以下示例展示了基本元素。有关更多详细信息，请参阅 crate 文档。

将 aws-sigv4 和 http crate 添加到 Cargo.toml 文件中：

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

源文件：

```
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);
```

处理 Rust AWS 开发工具包中的错误

了解 适用于 Rust 的 AWS SDK 返回错误的方式和时间对于使用 SDK 构建高质量的应用程序非常重要。以下各节介绍了您在 SDK 中可能遇到的各种错误以及如何正确处理它们。

每个操作都会返回一个错误类型设置为 SdkError<E, R = HttpResponse> 的 Result 类型。SdkError 是一个包含几种可能类型的枚举，称为变体。

服务错误

最常见的错误类型是 SdkError::ServiceError。该错误是指来自 AWS 服务的错误响应。例如，如果您尝试从 Amazon S3 获取不存在的对象，则 Amazon S3 会返回错误响应。

当您遇到时SdkError::ServiceError，表示您的请求已成功发送到， AWS 服务 但无法处理。这可能是因为请求的参数中存在错误，或者是因为服务端的问题。

错误变体中包含错误响应详细信息。以下示例展示了如何便捷地获取底层 ServiceError 变体并处理不同的错误情况：

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

错误元数据

每个服务错误都有额外的元数据，可以通过导入特定于服务的特性来访问这些元数据。

- *<service>*::error::ProvideErrorMetadata 特性提供了对服务返回的任何可用底层原始错误代码和错误消息的访问权限。
  - 对于 Amazon S3，这个特性是 aws_sdk_s3::error::ProvideErrorMetadata。

您还可以获取一些对服务错误故障排除可能有用的信息：

- *<service>*::operation::RequestId该特征添加了用于检索服务生成的唯一 AWS 请求 ID 的扩展方法。
  - 对于 Amazon S3，这个特性是 aws_sdk_s3::operation::RequestId。
- *<service>*::operation::RequestIdExt 特性添加了 extended_request_id() 方法，用于获取额外的扩展请求 ID。
  - 仅部分服务支持。
  - 对于 Amazon S3，这个特性是 aws_sdk_s3::operation::RequestIdExt。

使用 DisplayErrorContext 打印详细的错误

SDK 中的错误通常是由一连串失败造成的，例如：

1. 连接器返回了一个错误，分派请求失败。
2. 连接器返回了一个错误，因为凭证提供程序返回了一个错误。
3. 凭证提供程序返回了一个错误，因为它调用了一个服务，而该服务返回了一个错误。
4. 该服务返回了一个错误，因为凭证请求没有正确的授权。

默认情况下，显示此错误仅输出“分派失败”。这缺少有助于解决错误的详细信息。适用于 Rust 的 SDK 提供了一个名为 DisplayErrorContext 的简单错误报告程序。

- *<service>*::error::DisplayErrorContext 结构增加了输出完整错误上下文的功能。
  - 对于 Amazon S3，这个结构是 aws_sdk_s3::error::DisplayErrorContext。

当我们封装要显示的错误并打印出来时，DisplayErrorContext 会提供更详细的消息，类似于以下内容：

```
dispatch failure: other: Session token not found or invalid.
DispatchFailure(
    DispatchFailure {
        source: ConnectorError {
            kind: Other(None),
            source: ProviderError(
                ProviderError {
                    source: ProviderError(
                        ProviderError {
                            source: ServiceError(
                                ServiceError {
                                    source: UnauthorizedException(
                                        UnauthorizedException {
                                            message: Some("Session token not found or
 invalid"),
                                            meta: ErrorMetadata {
                                                code: Some("UnauthorizedException"),
                                                message: Some("Session token not found
 or invalid"),
                                                extras: Some({"aws_request_id":
 "1b6d7476-f5ec-4a16-9890-7684ccee7d01"})
                                            }
                                        }
                                    ),
                                    raw: Response {
                                        status: StatusCode(401),
                                        headers: Headers {
                                            headers: {
                                                "date": HeaderValue { _private:
 H0("Thu, 04 Jul 2024 07:41:21 GMT") },
                                                "content-type": HeaderValue { _private:
 H0("application/json") },
                                                "content-length": HeaderValue
 { _private: H0("114") },
                                                "access-control-expose-headers":
 HeaderValue { _private: H0("RequestId") },
```

```
                                                "access-control-expose-headers":
 HeaderValue { _private: H0("x-amzn-RequestId") },
                                                "requestid": HeaderValue { _private:
 H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") },
                                                "server": HeaderValue { _private:
 H0("AWS SSO") },
                                                "x-amzn-requestid": HeaderValue
 { _private: H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") }
                                            }
                                        },
                                        body: SdkBody {
                                            inner: Once(
                                                Some(
                                                    b"{
                                                        \"message\":\"Session token not
 found or invalid\",
                                                        \"__type\":
\"com.amazonaws.switchboard.portal#UnauthorizedException\"}"
                                                    )
                                                ),
                                            retryable: true
                                        },
                                        extensions: Extensions {
                                            extensions_02x: Extensions,
                                            extensions_1x: Extensions
                                        }
                                    }
                                }
                            )
                        }
                    )
                }
            ),
            connection: Unknown
        }
    }
)
```

在适用于 Rust 的 AWS SDK 中使用分页结果

当有效载荷太大而无法在单个响应中返回时，很多 AWS 操作都会返回截断的结果。相反，该服务会返回一部分数据和一个用于检索下一组项目的令牌。这种模式称为分页。

适用于 Rust 的 AWS SDK 包含操作生成器上的扩展方法 `into_paginator`，可用于自动对结果进行分页。您只需编写处理结果的代码。所有分页操作生成器都有一种可用的 `into_paginator()` 方法，该方法会公开一个 `PaginationStream<Item>` 来对结果进行分页。

- 在 Amazon S3 中，这种方法的一个示例是 `aws_sdk_s3::operation::list_objects_v2::builders::ListObjectsV2FluentBuilder::`

以下示例使用的是 Amazon Simple Storage Service。但是，对于任何具有一个或多个分页 API 的服务来说，这些概念都是相同的。

以下代码示例是使用 `try_collect()` 方法将所有分页结果收集到 `Vec` 的最简单示例：

```
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<_>>();
```

有时，您想要更好地控制分页，而不是同时将所有内容全部加载到内存中。以下示例遍历 Amazon S3 存储桶中的所有对象。

```
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);
    }
}
```

向适用于 Rust 的 AWS SDK 应用程序添加单元测试

虽然在 适用于 Rust 的 AWS SDK 项目中实施单元测试的方法有很多，但我们推荐以下几种：

- 使用 mockall 进行单元测试 – 使用 mockall crate 中的 automock 自动生成和执行测试。
- 静态重播 – 使用 AWS Smithy 运行时的 StaticReplayClient 创建一个虚假 HTTP 客户端，该客户端可以代替 AWS 服务通常使用的标准 HTTP 客户端。该客户端返回您指定的 HTTP 响应，而不是通过网络与服务通信，以便测试能够获得已知数据用于测试目的。
- 使用 aws-smithy-mocks 进行单元测试 – 使用 mock 和 aws-smithy-mocks crate 的 mock_client 来模拟 AWS SDK 客户端的响应，并创建 mock 规则来定义 SDK 应如何响应特定请求。

使用**mockall**适用于 Rust 的 AWS SDK 自动生成模拟

适用于 Rust 的 AWS SDK 提供了多种方法来测试与 AWS 服务之交互的代码。您可以使用 mockall crate 中流行的 automock 来自动生成测试所需的大多数 mock 实现。

此示例测试一个名为 determine_prefix_file_size() 的自定义方法。此方法调用一个调用 Amazon S3 的自定义 list_objects() 包装器方法。通过模拟 list_objects()，无需实际联系 Amazon S3 即可测试 determine_prefix_file_size() 方法。

1. 在项目目录的命令提示符中，将 mockall crate 添加为依赖项：

```
$ cargo add --dev mockall
```

使用 `--dev` 选项可将 crate 添加到 `Cargo.toml` 文件的 `[dev-dependencies]` 部分。作为开发依赖项，它不会被编译并包含在用于生产代码的最终二进制文件中。

此示例代码还使用 Amazon Simple Storage Service 作为示例 AWS 服务。

```
$ cargo add aws-sdk-s3
```

这会将 crate 添加到 `Cargo.toml` 文件的 `[dependencies]` 部分。

2. 添加 `mockall` crate 中的 `automock` 模块。

还应包括与您正在测试的 AWS 服务 相关的任何其他库，在本例中为 Amazon S3。

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

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

3. 接下来，添加代码，确定要使用应用程序的 Amazon S3 包装器结构的两个实现中的哪一个。

- 用于通过网络访问 Amazon S3 的真正实现。
- `mockall` 生成的 mock 实现。

在这个示例中，选中的一个名称为 `S3`。选择是基于 `test` 属性的条件进行的：

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

4. 该`S3Impl`结构是实际向发送请求的 Amazon S3 包装器结构的实现。 AWS

- 启用测试后，将不使用此代码，因为请求已发送到 mock 而不是 AWS。`dead_code` 属性告诉 linter，如果未使用 `S3Impl` 类型，不要报告问题。
- 条件 `#[cfg_attr(test, automock)]` 表示启用测试后，应设置 `automock` 属性。这会告诉 `mockall` 生成一个名为 `Mock`*`S3Impl`* 的 `S3Impl` mock。
- 在此示例中，`list_objects()` 方法是您要模拟的调用。`automock` 将自动为您创建 `expect_`*`list_objects()`* 方法。

```
#[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. 在名为 `test` 的模块中创建测试函数。

   - 条件 `#[cfg(test)]` 表示，如果 `test` 属性是 `true`，则 `mockall` 应构建测试模块。

```
#[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);
    }
}
```

- 每个测试都使用 `let mut mock = MockS3Impl::default();` 来创建 MockS3Impl 的 mock 实例。
- 它使用 mock 的 `expect_list_objects()` 方法（由 automock 自动创建）来设置当在代码的其他地方使用 `list_objects()` 方法时的预期结果。
- 在设置预期结果后，系统会通过调用 `determine_prefix_file_size()` 使用这些结果来测试函数。使用断言检查返回值以确认其是否正确。

6. `determine_prefix_file_size()` 函数使用 Amazon S3 包装器来获取前缀文件的大小：

```
#[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)
}
```

类型 `S3` 用于调用封装的适用于 Rust 的 SDK 函数，以便在发出 HTTP 请求时同时支持 `S3Impl` 和 `MockS3Impl`。启用测试后，由 `mockall` 自动生成的 mock 会报告任何测试失败的情况。

您可以在上查看这些示例的完整代码 GitHub。

在适用于 Rust 的 AWS SDK 中使用静态重播模拟 HTTP 流量

适用于 Rust 的 AWS SDK 提供了多种方法来测试与 AWS 服务之交互的代码。本主题介绍了如何使用 `StaticReplayClient` 创建虚假 HTTP 客户端，该客户端可以代替 AWS 服务通常使用的标准 HTTP 客户端。该客户端返回您指定的 HTTP 响应，而不是通过网络与服务通信，以便测试能够获得已知数据用于测试目的。

`aws-smithy-http-client` crate 包含一个名为 `StaticReplayClient` 的测试实用程序类。创建 AWS 服务 对象时，可以指定此 HTTP 客户端类而不是默认的 HTTP 客户端。

初始化 `StaticReplayClient` 时，您可以提供 HTTP 请求和响应对的列表作为 `ReplayEvent` 对象。在测试运行时，系统会记录每个 HTTP 请求，客户端将在事件列表中下一个 `ReplayEvent` 中找到的下一个 HTTP 响应作为 HTTP 客户端的响应返回。这样，测试就可以在没有网络连接的情况下使用已知数据运行。

使用静态重播

要使用静态重播，无需使用包装器。但是，您需要确定测试将使用的数据的实际网络流量应是什么样的，并将该流量数据提供给 `StaticReplayClient`，以便在每次 SDK 从 AWS 服务 客户端发出请求时使用。

> **Note**
>
> 有多种方法可以收集预期的网络流量，包括许多网络流量分析器和数据包嗅探器工具。 AWS CLI

- 创建 `ReplayEvent` 对象列表，指定预期的 HTTP 请求以及应针对这些请求返回的响应。
- 使用上一步创建的 HTTP 事务列表创建 `StaticReplayClient`。
- 为 AWS 客户端创建配置对象，将指定`StaticReplayClient`为该`Config`对象的`http_client`。
- 使用在上一步中创建的配置创建 AWS 服务 客户端对象。
- 使用配置为使用 `StaticReplayClient` 的服务对象，执行您要测试的操作。每当 SDK 向发送 API 请求时 AWS，都会使用列表中的下一个响应。

  > **Note**
  >
  > 即使发送的请求与 `ReplayEvent` 对象向量中的请求不匹配，系统也总是会返回列表中的下一个响应。

- 发出所有所需请求后，调用 `StaticReplayClient.assert_requests_match()` 函数以验证 SDK 发送的请求是否与 `ReplayEvent` 对象列表中的请求相匹配。

示例

我们来看上一个示例中同一 `determine_prefix_file_size()` 函数的测试，但这次使用的是静态重播而不是模拟。

1. 在项目目录的命令提示符中，将 `aws-smithy-http-client` crate 添加为依赖项：

   ```
   $ cargo add --dev aws-smithy-http-client --features test-util
   ```

   使用 `--dev` 选项可将 crate 添加到 `Cargo.toml` 文件的 `[dev-dependencies]` 部分。作为开发依赖项，它不会被编译并包含在用于生产代码的最终二进制文件中。

   此示例代码还使用 Amazon Simple Storage Service 作为示例 AWS 服务。

   ```
   $ cargo add aws-sdk-s3
   ```

这会将 crate 添加到 Cargo.toml 文件的 [dependencies] 部分。

2. 在测试代码模块中，包含您需要的两种类型。

```
use aws_smithy_http_client::test_util::{ReplayEvent, StaticReplayClient};
use aws_sdk_s3::primitives::SdkBody;
```

3. 测试首先创建表示测试期间应发生的每个 HTTP 事务的 ReplayEvent 结构。每个事件都包含一个 HTTP 请求对象和一个 HTTP 响应对象，表示 AWS 服务 通常会回复的信息。这些事件会传递到对 StaticReplayClient::new() 的调用中：

```
        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]);
```

结果存储在 replay_client 中。这表示一个 HTTP 客户端，适用于 Rust 的 SDK 可以通过在客户端配置中指定该客户端来使用它。

4. 要创建 Amazon S3 客户端，请使用配置对象调用客户端类的 `from_conf()` 函数来创建客户端：

```
        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(),
        );
```

使用生成器的 `http_client()` 方法指定配置对象，使用 `credentials_provider()` 方法指定凭证。凭证是使用名为 `make_s3_test_credentials()` 的函数创建的，该函数会返回一个虚假凭证结构：

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

这些凭证不一定是有效的，因为它们实际上不会被发送到 AWS。

5. 通过调用需要测试的函数来运行测试。在此示例中，该函数名称为 `determine_prefix_file_size()`。它的第一个参数是用于其请求的 Amazon S3 客户端对象。因此，请指定使用 `StaticReplayClient` 创建的客户端，以便请求由该客户端处理，而不是通过网络传出：

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

        assert_eq!(19, size);

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

对 `determine_prefix_file_size()` 的调用完成后，系统会使用断言来确认返回的值是否与预期值匹配。然后，调用 `StaticReplayClient` 方法 `assert_requests_match()` 函数。此函数扫描记录的 HTTP 请求，并确认它们是否与创建重播客户端时提供的 `ReplayEvent` 对象数组中指定的请求相匹配。

您可以在上查看这些示例的完整代码 GitHub。

在适用于 Rust 的 AWS SDK **`aws-smithy-mocks`** 中进行单元测试

适用于 Rust 的 AWS SDK 提供了多种方法来测试与 AWS 服务之交互的代码。本主题介绍了如何使用 `aws-smithy-mocks` crate，它提供了一种简单而强大的方法来模拟 AWS SDK 客户端响应以用于测试目的。

概述

在为使用的代码编写测试时 AWS 服务，您通常希望避免进行实际的网络调用。`aws-smithy-mocks` crate 提供了一种解决方案，支持：

- 创建 mock 规则，用于定义 SDK 应如何响应特定请求。
- 返回不同类型的响应（成功、错误、HTTP 响应）。
- 根据请求的属性匹配请求。
- 定义用于测试重试行为的响应序列。
- 验证规则是否按预期使用。

添加依赖项

在项目目录的命令提示符中，将 `aws-smithy-mocks` crate 添加为依赖项：

```
$ cargo add --dev aws-smithy-mocks
```

使用 `--dev` 选项可将 crate 添加到 `Cargo.toml` 文件的 `[dev-dependencies]` 部分。作为开发依赖项，它不会被编译并包含在用于生产代码的最终二进制文件中。

此示例代码还使用 Amazon 简单存储服务作为示例 AWS 服务，并且需要功能`test-util`。

```
$ cargo add aws-sdk-s3 --features test-util
```

这会将 crate 添加到 `Cargo.toml` 文件的 `[dependencies]` 部分。

基本用法

以下是一个简单的示例，展示了如何使用 `aws-smithy-mocks` 测试与 Amazon Simple Storage Service（Amazon S3）交互的代码：

```
use aws_sdk_s3::operation::get_object::GetObjectOutput;
use aws_sdk_s3::primitives::ByteStream;
use aws_smithy_mocks::{mock, mock_client};

#[tokio::test]
async fn test_s3_get_object() {
    // Create a rule that returns a successful response
    let get_object_rule = mock!(aws_sdk_s3::Client::get_object)
        .then_output(|| {
            GetObjectOutput::builder()
                .body(ByteStream::from_static(b"test-content"))
                .build()
        });

    // Create a mocked client with the rule
    let s3 = mock_client!(aws_sdk_s3, [&get_object_rule]);

    // Use the client as you would normally
    let result = s3
        .get_object()
        .bucket("test-bucket")
        .key("test-key")
        .send()
        .await
        .expect("success response");

    // Verify the response
    let data = result.body.collect().await.expect("successful read").to_vec();
    assert_eq!(data, b"test-content");

    // Verify the rule was used
    assert_eq!(get_object_rule.num_calls(), 1);
}
```

创建 mock 规则

规则是使用 mock! 宏创建的，该宏将客户端操作作为参数。然后，您可以配置规则的行为方式。

匹配请求

您可以通过匹配请求属性来使规则更加具体：

```
let rule = mock!(Client::get_object)
    .match_requests(|req| req.bucket() == Some("test-bucket") && req.key() ==
 Some("test-key"))
    .then_output(|| {
        GetObjectOutput::builder()
            .body(ByteStream::from_static(b"test-content"))
            .build()
    });
```

不同的响应类型

您可以返回不同类型的响应：

```
// Return a successful response
let success_rule = mock!(Client::get_object)
    .then_output(|| GetObjectOutput::builder().build());

// Return an error
let error_rule = mock!(Client::get_object)
    .then_error(|| GetObjectError::NoSuchKey(NoSuchKey::builder().build()));

// Return a specific HTTP response
let http_rule = mock!(Client::get_object)
    .then_http_response(|| {
        HttpResponse::new(
            StatusCode::try_from(503).unwrap(),
            SdkBody::from("service unavailable")
        )
    });
```

测试重试行为

aws-smithy-mocks 最强大的功能之一是能够通过定义响应序列来测试重试行为：

```
// Create a rule that returns 503 twice, then succeeds
```

```
let retry_rule = mock!(aws_sdk_s3::Client::get_object)
    .sequence()
    .http_status(503, None)                          // First call returns 503
    .http_status(503, None)                          // Second call returns 503
    .output(|| GetObjectOutput::builder().build())   // Third call succeeds
    .build();

// With repetition using times()
let retry_rule = mock!(Client::get_object)
    .sequence()
    .http_status(503, None)
    .times(2)                                        // First two calls return 503
    .output(|| GetObjectOutput::builder().build())   // Third call succeeds
    .build();
```

规则模式

您可以使用规则模式控制规则的匹配和应用方式：

```
// Sequential mode: Rules are tried in order, and when a rule is exhausted, the next
 rule is used
let client = mock_client!(aws_sdk_s3, RuleMode::Sequential, [&rule1, &rule2]);

// MatchAny mode: The first matching rule is used, regardless of order
let client = mock_client!(aws_sdk_s3, RuleMode::MatchAny, [&rule1, &rule2]);
```

示例：测试重试行为

以下是一个更完整的示例，展示了如何测试重试行为：

```
use aws_sdk_s3::operation::get_object::GetObjectOutput;
use aws_sdk_s3::config::RetryConfig;
use aws_sdk_s3::primitives::ByteStream;
use aws_smithy_mocks::{mock, mock_client, RuleMode};

#[tokio::test]
async fn test_retry_behavior() {
    // Create a rule that returns 503 twice, then succeeds
    let retry_rule = mock!(aws_sdk_s3::Client::get_object)
        .sequence()
        .http_status(503, None)
        .times(2)
        .output(|| GetObjectOutput::builder()
```

```
            .body(ByteStream::from_static(b"success"))
            .build())
        .build();

    // Create a mocked client with the rule and custom retry configuration
    let s3 = mock_client!(
        aws_sdk_s3,
        RuleMode::Sequential,
        [&retry_rule],
        |client_builder| {
            client_builder.retry_config(RetryConfig::standard().with_max_attempts(3))
        }
    );

    // This should succeed after two retries
    let result = s3
        .get_object()
        .bucket("test-bucket")
        .key("test-key")
        .send()
        .await
        .expect("success after retries");

    // Verify the response
    let data = result.body.collect().await.expect("successful read").to_vec();
    assert_eq!(data, b"success");

    // Verify all responses were used
    assert_eq!(retry_rule.num_calls(), 3);
}
```

示例：基于请求参数的不同响应

您还可以创建规则，根据请求参数返回不同的响应：

```
use aws_sdk_s3::operation::get_object::{GetObjectOutput, GetObjectError};
use aws_sdk_s3::types::error::NoSuchKey;
use aws_sdk_s3::Client;
use aws_sdk_s3::primitives::ByteStream;
use aws_smithy_mocks::{mock, mock_client, RuleMode};

#[tokio::test]
async fn test_different_responses() {
```

```
    // Create rules for different request parameters
    let exists_rule = mock!(Client::get_object)
        .match_requests(|req| req.bucket() == Some("test-bucket") && req.key() ==
Some("exists"))
        .sequence()
        .output(|| GetObjectOutput::builder()
            .body(ByteStream::from_static(b"found"))
            .build())
        .build();

    let not_exists_rule = mock!(Client::get_object)
        .match_requests(|req| req.bucket() == Some("test-bucket") && req.key() ==
Some("not-exists"))
        .sequence()
        .error(|| GetObjectError::NoSuchKey(NoSuchKey::builder().build()))
        .build();

    // Create a mocked client with the rules in MatchAny mode
    let s3 = mock_client!(aws_sdk_s3, RuleMode::MatchAny, [&exists_rule,
&not_exists_rule]);

    // Test the "exists" case
    let result1 = s3
        .get_object()
        .bucket("test-bucket")
        .key("exists")
        .send()
        .await
        .expect("object exists");

    let data = result1.body.collect().await.expect("successful read").to_vec();
    assert_eq!(data, b"found");

    // Test the "not-exists" case
    let result2 = s3
        .get_object()
        .bucket("test-bucket")
        .key("not-exists")
        .send()
        .await;

    assert!(result2.is_err());
    assert!(matches!(result2.unwrap_err().into_service_error(),
                    GetObjectError::NoSuchKey(_)));
```

```
}
```

最佳实践

使用 `aws-smithy-mocks` 进行测试时：

1. 匹配特定请求：使用 `match_requests()` 确保规则仅应用于预期请求，尤其是使用 `RuleMode::MatchAny`。
2. 验证规则使用情况：检查 `rule.num_calls()` 以确保规则已实际使用。
3. 测试错误处理：创建返回错误的规则，以测试代码是如何处理故障的。
4. 测试重试逻辑：使用响应序列来验证代码是否正确处理了任何自定义重试分类器或其他重试行为。
5. 保持测试的针对性：针对不同的场景创建单独的测试，而不是使用一个测试来测试所有内容。

在适用于 Rust 的 AWS SDK 中使用等待器

等待器是一种客户端抽象，用于轮询资源，直到达到所需状态，或者直到确定资源不会进入所需状态。在使用最终一致的服务（例如 Amazon Simple Storage Service）或异步创建资源的服务（例如 Amazon Elastic Compute Cloud）时，这是一项常见的任务。编写逻辑以持续轮询资源状态可能非常繁琐且容易出错。等待器的目标是将这项责任从客户代码转移到适用于 Rust 的 AWS SDK，后者对 AWS 操作时机方面有深入了解。

AWS 服务为包含 *`<service>`*`::waiters` 模块的等待器提供支持。

- *`<service>`*`::client::Waiters` 特性为客户端提供等待器方法。这些方法是为 `Client` 结构体实现的。所有等待器方法都遵循 `wait_until_`*`<Condition>`* 的标准命名约定
  - 对于 Amazon S3，这个特性是 `aws_sdk_s3::client::Waiters`。

以下示例使用的是 Amazon S3。但是，对于任何定义一个或多个等待器的 AWS 服务，概念都是一样的。

以下代码示例演示了如何使用等待器函数而不是编写轮询逻辑来等待存储桶在创建后存在。

```
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**
>
> 每个 wait 方法都会返回一个 `Result<FinalPoll<...>, WaiterError<...>>` ，可用于在达到所需条件或返回所需错误后获得最终响应。有关详细信息，请参阅 Rust API 文档中的 FinalPoll 和 WaiterError。

适用于 Rust 的 SDK 代码示例

本主题中的代码示例向您展示了如何使用适用于 Rust 的 AWS SDK AWS。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务服务结合来完成特定任务的代码示例。

某些服务包含其他示例类别，这些类别说明如何利用特定于服务的库或函数。

Services

- 使用 SDK for Rust 的 API Gateway 示例
- 使用 SDK for Rust 的 API 网关管理 API 示例
- 使用 SDK for Rust 的 Application Auto Scaling 示例
- 使用 SDK for Rust 的 Aurora 示例
- 使用 SDK for Rust 的 Auto Scaling 示例
- 使用适用于 Rust 的 SDK 的 Amazon Bedrock 运行时示例
- 使用适用于 Rust 的 SDK 的 Amazon Bedrock 代理运行时示例
- 使用 SDK for Rust 的 Amazon Cognito 身份提供者的代码示例
- 使用 SDK for Rust 的 Amazon Cognito Sync 示例
- 使用适用于 Rust 的 SDK 的 Firehose 示例
- 使用适用于 Rust 的 SDK 的 Amazon DocumentDB 示例
- 使用 SDK for Rust 的 DynamoDB 示例
- 使用 SDK for Rust 的 Amazon EBS 示例
- 使用适用于 Rust 的软件开发工具包的亚马逊 EC2 示例
- 使用 SDK for Rust 的 Amazon ECR 示例
- 使用 SDK for Rust 的 Amazon ECS 示例
- 使用 SDK for Rust 的 Amazon EKS 示例
- AWS Glue 使用 Rust 版 SDK 的示例

- 使用 SDK for Rust 的 IAM 示例
- AWS IoT 使用 Rust 版 SDK 的示例
- 使用 SDK for Rust 的 Kinesis 示例
- AWS KMS 使用 Rust 版 SDK 的示例
- 使用 SDK for Rust 的 Lambda 示例
- MediaLive 使用 Rust 版 SDK 的示例
- MediaPackage 使用 Rust 版 SDK 的示例
- 使用适用于 Rust 的 SDK 的 Amazon MSK 示例
- 使用 SDK for Rust 的 Amazon Polly 示例
- 使用适用于 Rust 的 SDK 的 Amazon RDS 示例
- 使用 SDK for Rust 的 Amazon RDS 数据服务示例
- 使用适用于 Rust 的 SDK 的 Amazon Rekognition 示例
- 使用 SDK for Rust 的 Route 53 示例
- 使用 SDK for Rust 的 Amazon S3 示例
- SageMaker 使用 Rust 版 SDK 的人工智能示例
- 使用 SDK for Rust 的 Secrets Manager 示例
- 使用 SDK for Rust 的 Amazon SES API v2 示例
- 使用 SDK for Rust 的 Amazon SNS 示例
- 使用 SDK for Rust 的 Amazon SQS 示例
- AWS STS 使用 Rust 版 SDK 的示例
- 使用 SDK for Rust 的 Systems Manager 示例
- 使用适用于 Rust 的 SDK 的 Amazon Transcribe 示例

使用 SDK for Rust 的 API Gateway 示例

以下代码示例向您展示了如何使用带有 API Gateway 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务服务结合来完成特定任务的代码示例。

AWS 社区贡献就是由多个团队创建和维护的示例 AWS。要提供反馈，请使用链接存储库中提供的机制。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 场景
- AWS 社区捐款

操作

`GetRestApis`

以下代码示例演示了如何使用 `GetRestApis`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

显示该地区的 Amazon API Gateway REST APIs 。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用GetRestApis于 Rust 的AWS SDK API 参考。

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 Rust 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

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

AWS 社区捐款

构建和测试无服务器应用程序

以下代码示例演示如何使用 API Gateway 与 Lambda 和 DynamoDB 构建和测试无服务器应用程序

适用于 Rust 的 SDK

演示如何使用 Rust SDK 构建和测试包含 API Gateway 以及 Lambda 和 DynamoDB 的无服务器应用程序。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda

使用 SDK for Rust 的 API 网关管理 API 示例

以下代码示例向您展示了如何使用带有 API Gateway Management API 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`PostToConnection`

以下代码示例演示了如何使用 `PostToConnection`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
```

- 有关 API 的详细信息，请参阅适用PostToConnection于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Application Auto Scaling 示例

以下代码示例向您展示了如何使用带有 Application Auto Scaling 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`DescribeScalingPolicies`

以下代码示例演示了如何使用 `DescribeScalingPolicies`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeScalingPolicies于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Aurora 示例

以下代码示例向您展示了如何使用带有 Aurora 的 Rust AWS 开发工具包来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

开始使用 Aurora

以下代码示例显示如何开始使用 Aurora。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅DBClusters在 AWS SDK for Rust API 参考中描述。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建自定义 Aurora 数据库集群参数组并设置参数值。
- 创建一个使用参数组的数据库集群。
- 创建包含数据库的数据库实例。
- 拍摄数据库集群的快照，然后清理资源。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

包含适用于 Aurora 场景的特定场景函数的库。

```
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;
```

围绕 RDS 客户端包装器使用自动模拟来对该库进行测试。

```
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");
}
```

一个从前端到末端运行场景的二进制文件，使用查询器，以便用户可以做出一些决定。

```
use std::fmt::Display;

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

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

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

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

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

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

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)
        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
}

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

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

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

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

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

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
                "Invalid instance class selection",
            )?;
            scenario.set_instance_class(Some(instance_class))
        }
        Err(err) => return Err(anyhow!("Failed to get instance classes for engine:
 {err}")),
    }

    Ok(scenario)
}
```

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

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

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

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

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

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

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

    let password = inquire::Text::new("Password for the database (minimum 8
 characters)")
        .with_validator(|i: &str| {
```

```
            if i.len() >= 8 {
                Ok(inquire::validator::Validation::Valid)
            } else {
                Ok(inquire::validator::Validation::Invalid(
                    "Password must be at least 8 characters".into(),
                ))
            }
        })
        .prompt();

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

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

    Ok(())
}

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

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

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

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

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
```

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

    Ok(())
}

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

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

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

    // Clean up the instance, cluster, and parameter group, waiting for the instance
 and cluster to delete before moving on.
    let clean_up = scenario.clean_up().await;
    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
        }
    }

    if warnings.is_empty() {
        Ok(())
    } else {
```

```
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
    }
}

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

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

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

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

    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
```

```
            Err(error) => {
                warnings.push(
                    format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}
```

围绕 Amazon RDS 服务的包装器，允许自动对测试进行模拟。

```
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
        delete_db_cluster_parameter_group::{
            DeleteDBClusterParameterGroupError, DeleteDbClusterParameterGroupOutput,
        },
        delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
        describe_db_cluster_endpoints::{
            DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
        },
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
```

```
        },
        describe_db_clusters::{DescribeDBClustersError, DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

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

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

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

pub struct RdsImpl {
    pub inner: RdsClient,
}

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

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

```
            .send()
            .await
    }

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

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

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

```
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

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

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

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
```

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

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

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

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
```

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

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

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

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
```

```
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

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

在这种情况下使用的带依赖项的 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"] }
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的以下主题。
  - 创建DBCluster
  - 创建DBClusterParameterGroup
  - 创建DBCluster快照
  - 创建DBInstance
  - 删除DBCluster
  - 删除DBClusterParameterGroup
  - 删除DBInstance
  - 描述DBClusterParameterGroups
  - 描述DBCluster参数
  - 描述DBCluster快照
  - 描述DBClusters
  - 描述DBEngine版本
  - 描述DBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

操作

`CreateDBCluster`

以下代码示例演示了如何使用 `CreateDBCluster`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   // 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;
}
```

- 有关 API 的详细信息，请参阅DBCluster在 AWS SDK 中创建 for Rust API 参考。

`CreateDBClusterParameterGroup`

以下代码示例演示了如何使用 `CreateDBClusterParameterGroup`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅DBClusterParameterGroup在 AWS SDK 中创建 for Rust API 参考。

`CreateDBClusterSnapshot`

以下代码示例演示了如何使用 `CreateDBClusterSnapshot`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(), ScenarioError>
{
        if self.password.is_none() {
```

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

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

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

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

        let create_db_instance = self
            .rds
            .create_db_instance(
```

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

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

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

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

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
```

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

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

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

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

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

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

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

    pub async fn snapshot_cluster(
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

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

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

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

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

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

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

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

```
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

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

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

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

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

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

```
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

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

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

```
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

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

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

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

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

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

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

- 有关 API 的详细信息，请参阅在 AWS SDK 中创建DBCluster快照 for Rust API 参考。

`CreateDBInstance`

以下代码示例演示了如何使用 `CreateDBInstance`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   // 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;
}
```

- 有关 API 的详细信息，请参阅DBInstance在 AWS SDK 中创建 for Rust API 参考。

`DeleteDBCluster`

以下代码示例演示了如何使用 `DeleteDBCluster`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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;
}
```

- 有关 API 的详细信息，请参阅DBCluster在 AWS SDK for Rust 中删除 API 参考。

`DeleteDBClusterParameterGroup`

以下代码示例演示了如何使用 `DeleteDBClusterParameterGroup`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

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

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

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

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

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

```
                }
            }
        }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- 有关 API 的详细信息，请参阅DBClusterParameterGroup在 AWS SDK for Rust 中删除 API 参考。

`DeleteDBInstance`

以下代码示例演示了如何使用 `DeleteDBInstance`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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;
}
```

- 有关 API 的详细信息，请参阅DBInstance在 AWS SDK for Rust 中删除 API 参考。

`DescribeDBClusterParameters`

以下代码示例演示了如何使用 `DescribeDBClusterParameters`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅在 AWS SDK 中描述DBCluster参数 for Rust API 参考。

`DescribeDBClusters`

以下代码示例演示了如何使用 `DescribeDBClusters`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    // 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;
}
```

- 有关 API 的详细信息，请参阅DBClusters在 AWS SDK for Rust API 参考中描述。

`DescribeDBEngineVersions`

以下代码示例演示了如何使用 `DescribeDBEngineVersions`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅《描述适用于 Rust 的AWS SDK 中的DBEngine版本 API 参考》。

`DescribeDBInstances`

以下代码示例演示了如何使用 `DescribeDBInstances`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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;
}
```

- 有关 API 的详细信息，请参阅DBInstances在 AWS SDK for Rust API 参考中描述。

`DescribeOrderableDBInstanceOptions`

以下代码示例演示了如何使用 `DescribeOrderableDBInstanceOptions`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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"
    );
}
```

- 有关 API 的详细信息，请参阅适用于 Rust AWS 的 SDK 中的DescribeOrderableDBInstance选项 API 参考。

`ModifyDBClusterParameterGroup`

以下代码示例演示了如何使用 `ModifyDBClusterParameterGroup`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅DBClusterParameterGroup在 AWS SDK for Rust 中修改 API 参考。

使用 SDK for Rust 的 Auto Scaling 示例

以下代码示例向您展示了如何使用带有 Auto Scaling 的 Rust AWS 开发工具包来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

开始使用 Auto Scaling

以下代码示例显示了如何开始使用 Auto Scaling。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
```

```
}
```

- 有关 API 的详细信息，请参阅适用DescribeAutoScalingGroups于 Rust 的AWS SDK API 参考。

基本功能

了解基本功能

以下代码示例展示了如何：

- 使用启动模板和可用区创建一个 Amazon A EC2 uto Scaling 群组，并获取有关正在运行的实例的信息。
- 启用 Amazon CloudWatch 指标收集。
- 更新组的所需容量，并等待实例启动。
- 终止组中的实例。
- 列出为响应用户请求和容量变化而发生的扩缩活动。
- 获取 CloudWatch 指标的统计数据，然后清理资源。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

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

[dependencies]
```

```
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-autoscaling = { version = "1.3.0" }
aws-sdk-ec2 = { version = "1.3.0" }
aws-types = { version = "1.0.1" }
tokio = { version = "1.20.1", features = ["full"] }
clap = { version = "4.4", features = ["derive"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
anyhow = "1.0.75"
tracing = "0.1.37"
tokio-stream = "0.1.14"


use std::{collections::BTreeSet, fmt::Display};

use anyhow::anyhow;
use autoscaling_code_examples::scenario::{AutoScalingScenario, ScenarioError};
use tracing::{info, warn};

async fn show_scenario_description(scenario: &AutoScalingScenario, event: &str) {
    let description = scenario.describe_scenario().await;
    info!("DescribeAutoScalingInstances: {event}\n{description}");
}

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

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

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

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

```
        Ok(())
    }
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();

    let shared_config = aws_config::from_env().load().await;

    let mut warnings = Warnings::default();

    // 1. Create an EC2 launch template that you'll use to create an auto scaling
 group. Bonus: use SDK with EC2.CreateLaunchTemplate to create the launch template.
    // 2. CreateAutoScalingGroup: pass it the launch template you created in step 0.
 Give it min/max of 1 instance.
    // 4. EnableMetricsCollection: enable all metrics or a subset.
    let scenario = match AutoScalingScenario::prepare_scenario(&shared_config).await
 {
        Ok(scenario) => scenario,
        Err(errs) => {
            let err_str = errs
                .into_iter()
                .map(|e| e.to_string())
                .collect::<Vec<String>>()
                .join(", ");
            return Err(anyhow!("Failed to initialize scenario: {err_str}"));
        }
    };

    info!("Prepared autoscaling scenario:\n{scenario}");

    let stable = scenario.wait_for_stable(1).await;
    if let Err(err) = stable {
        warnings.push(
            "There was a problem while waiting for group to be stable",
            err,
        );
    }

    // 3. DescribeAutoScalingInstances: show that one instance has launched.
    show_scenario_description(
        &scenario,
        "show that the group was created and one instance has launched",
```

```
    )
    .await;

    // 5. UpdateAutoScalingGroup: update max size to 3.
    let scale_max_size = scenario.scale_max_size(3).await;
    if let Err(err) = scale_max_size {
        warnings.push("There was a problem scaling max size", err);
    }

    // 6. DescribeAutoScalingGroups: the current state of the group
    show_scenario_description(
        &scenario,
        "show the current state of the group after setting max size",
    )
    .await;

    // 7. SetDesiredCapacity: set desired capacity to 2.
    let scale_desired_capacity = scenario.scale_desired_capacity(2).await;
    if let Err(err) = scale_desired_capacity {
        warnings.push("There was a problem setting desired capacity", err);
    }

    //   Wait for a second instance to launch.
    let stable = scenario.wait_for_stable(2).await;
    if let Err(err) = stable {
        warnings.push(
            "There was a problem while waiting for group to be stable",
            err,
        );
    }

    // 8. DescribeAutoScalingInstances: show that two instances are launched.
    show_scenario_description(
        &scenario,
        "show that two instances are launched after setting desired capacity",
    )
    .await;

    let ids_before = scenario
        .list_instances()
        .await
        .map(|v| v.into_iter().collect::<BTreeSet<_>>())
        .unwrap_or_default();
```

```
    // 9. TerminateInstanceInAutoScalingGroup: terminate one of the instances in the
group.
    let terminate_some_instance = scenario.terminate_some_instance().await;
    if let Err(err) = terminate_some_instance {
        warnings.push("There was a problem replacing an instance", err);
    }

    let wait_after_terminate = scenario.wait_for_stable(1).await;
    if let Err(err) = wait_after_terminate {
        warnings.push(
            "There was a problem waiting after terminating an instance",
            err,
        );
    }

    let wait_scale_up_after_terminate = scenario.wait_for_stable(2).await;
    if let Err(err) = wait_scale_up_after_terminate {
        warnings.push(
            "There was a problem waiting for scale up after terminating an
instance",
            err,
        );
    }

    let ids_after = scenario
        .list_instances()
        .await
        .map(|v| v.into_iter().collect::<BTreeSet<_>>())
        .unwrap_or_default();

    let difference = ids_after.intersection(&ids_before).count();
    if !(difference == 1 && ids_before.len() == 2 && ids_after.len() == 2) {
        warnings.push(
            "Before and after set not different",
            ScenarioError::with(format!("{difference}")),
        );
    }

    // 10. DescribeScalingActivities: list the scaling activities that have occurred
for the group so far.
    show_scenario_description(
        &scenario,
        "list the scaling activities that have occurred for the group so far",
    )
```

```
    .await;

    // 11. DisableMetricsCollection
    let scale_group = scenario.scale_group_to_zero().await;
    if let Err(err) = scale_group {
        warnings.push("There was a problem scaling the group to 0", err);
    }
    show_scenario_description(&scenario, "Scenario scaled to 0").await;

    // 12. DeleteAutoScalingGroup (to delete the group you must stop all instances):
    // 13. Delete LaunchTemplate.
    let clean_scenario = scenario.clean_scenario().await;
    if let Err(errs) = clean_scenario {
        for err in errs {
            warnings.push("There was a problem cleaning the scenario", err);
        }
    } else {
        info!("The scenario has been cleaned up!");
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        Err(anyhow!(
            "There were warnings during scenario execution:\n{warnings}"
        ))
    }
}

pub mod scenario;


use std::{
    error::Error,
    fmt::{Debug, Display},
    time::{Duration, SystemTime},
};

use anyhow::anyhow;
use aws_config::SdkConfig;
use aws_sdk_autoscaling::{
    error::{DisplayErrorContext, ProvideErrorMetadata},
    types::{Activity, AutoScalingGroup, LaunchTemplateSpecification},
};
```

```
use aws_sdk_ec2::types::RequestLaunchTemplateData;
use tracing::trace;

const LAUNCH_TEMPLATE_NAME: &str =
 "SDK_Code_Examples_EC2_Autoscaling_template_from_Rust_SDK";
const AUTOSCALING_GROUP_NAME: &str =
 "SDK_Code_Examples_EC2_Autoscaling_Group_from_Rust_SDK";
const MAX_WAIT: Duration = Duration::from_secs(5 * 60); // Wait at most 25 seconds.
const WAIT_TIME: Duration = Duration::from_millis(500); // Wait half a second at a
 time.

struct Waiter {
    start: SystemTime,
    max: Duration,
}

impl Waiter {
    fn new() -> Self {
        Waiter {
            start: SystemTime::now(),
            max: MAX_WAIT,
        }
    }

    async fn sleep(&self) -> Result<(), ScenarioError> {
        if SystemTime::now()
            .duration_since(self.start)
            .unwrap_or(Duration::MAX)
            > self.max
        {
            Err(ScenarioError::with(
                "Exceeded maximum wait duration for stable group",
            ))
        } else {
            tokio::time::sleep(WAIT_TIME).await;
            Ok(())
        }
    }
}

pub struct AutoScalingScenario {
    ec2: aws_sdk_ec2::Client,
    autoscaling: aws_sdk_autoscaling::Client,
    launch_template_arn: String,
```

```
    auto_scaling_group_name: String,
}

impl Display for AutoScalingScenario {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        f.write_fmt(format_args!(
            "\tLaunch Template ID: {}\n",
            self.launch_template_arn
        ))?;
        f.write_fmt(format_args!(
            "\tScaling Group Name: {}\n",
            self.auto_scaling_group_name
        ))?;

        Ok(())
    }
}

pub struct AutoScalingScenarioDescription {
    group: Result<Vec<String>, ScenarioError>,
    instances: Result<Vec<String>, anyhow::Error>,
    activities: Result<Vec<Activity>, anyhow::Error>,
}

impl Display for AutoScalingScenarioDescription {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "\t      Group status:")?;
        match &self.group {
            Ok(groups) => {
                for status in groups {
                    writeln!(f, "\t\t- {status}")?;
                }
            }
            Err(e) => writeln!(f, "\t\t! - {e}")?,
        }
        writeln!(f, "\t         Instances:")?;
        match &self.instances {
            Ok(instances) => {
                for instance in instances {
                    writeln!(f, "\t\t- {instance}")?;
                }
            }
            Err(e) => writeln!(f, "\t\t! {e}")?,
        }
```

```
        writeln!(f, "\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, "{display}")
    }
}

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

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

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

impl Error for ScenarioError {
    // While `Error` can capture `source` information about the underlying error,
 for this example
    // the ScenarioError captures the underlying information in MetadataError and
 treats it as a
    // single Error from this Crate. In other contexts, it may be appropriate to
 model the error
    // as including the SdkError as its source.
}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
```

```
}

impl AutoScalingScenario {
    pub async fn prepare_scenario(sdk_config: &SdkConfig) -> Result<Self,
 Vec<ScenarioError>> {
        let ec2 = aws_sdk_ec2::Client::new(sdk_config);
        let autoscaling = aws_sdk_autoscaling::Client::new(sdk_config);

        let auto_scaling_group_name = String::from(AUTOSCALING_GROUP_NAME);

        // Before creating any resources, prepare the list of AZs
        let availablity_zones = ec2.describe_availability_zones().send().await;
        if let Err(err) = availablity_zones {
            return Err(vec![ScenarioError::new("Failed to find AZs", &err)]);
        }

        let availability_zones: Vec<String> = availablity_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 retuning 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::<Result<Vec<_>, _>>()
            .await
            .map_err(|e| {
                anyhow!(
                    "There was an error retrieving scaling activities: {}",
                    DisplayErrorContext(&e)
                )
            });

        AutoScalingScenarioDescription {
            group,
            instances,
            activities,
        }
    }

    async fn get_group(&self) -> Result<AutoScalingGroup, ScenarioError> {
        let describe_auto_scaling_groups = self
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await;

        if let Err(err) = describe_auto_scaling_groups {
            return Err(ScenarioError::new(
                format!(
                    "Failed to get status of autoscaling group {}",
                    self.auto_scaling_group_name.clone()
                )
                .as_str(),
                &err,
            ));
        }

        let describe_auto_scaling_groups_output =
describe_auto_scaling_groups.unwrap();
```

```
        let auto_scaling_groups =
describe_auto_scaling_groups_output.auto_scaling_groups();
        let auto_scaling_group = auto_scaling_groups.first();

        if auto_scaling_group.is_none() {
            return Err(ScenarioError::with(format!(
                "Could not find autoscaling group {}",
                self.auto_scaling_group_name.clone()
            )));
        }

        Ok(auto_scaling_group.unwrap().clone())
    }

    pub async fn wait_for_no_scaling(&self) -> Result<(), ScenarioError> {
        let waiter = Waiter::new();
        let mut scaling = true;
        while scaling {
            waiter.sleep().await?;
            let describe_activities = self
                .autoscaling
                .describe_scaling_activities()
                .auto_scaling_group_name(self.auto_scaling_group_name.clone())
                .send()
                .await
                .map_err(|e| {
                    ScenarioError::new("Failed to get autoscaling activities for
group", &e)
                })?;
            let activities = describe_activities.activities();
            trace!(
                "Waiting for no scaling found {} activities",
                activities.len()
            );
            scaling = activities.iter().any(|a| a.progress() < Some(100));
        }
        Ok(())
    }

    pub async fn wait_for_stable(&self, size: usize) -> Result<(), ScenarioError> {
        self.wait_for_no_scaling().await?;

        let mut group = self.get_group().await?;
        let mut count = count_group_instances(&group);
```

```
        let waiter = Waiter::new();
        while count != size {
            trace!("Waiting for stable {size} (current: {count})");
            waiter.sleep().await?;
            group = self.get_group().await?;
            count = count_group_instances(&group);
        }

        Ok(())
    }

    pub async fn list_instances(&self) -> Result<Vec<String>, ScenarioError> {
        // The direct way to list instances is by using DescribeAutoScalingGroup's
instances property. However, this returns a Vec<Instance>, as opposed to a
Vec<AutoScalingInstanceDetails>.
        // Ok(self.get_group().await?.instances.unwrap_or_default().map(|i|
i.instance_id.clone().unwrap_or_default()).filter(|id| !id.is_empty()).collect())

        // Alternatively, and for the sake of example, DescribeAutoScalingInstances
returns a list that can be filtered by the client.
        self.autoscaling
            .describe_auto_scaling_instances()
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
            .map(|items| {
                items
                    .into_iter()
                    .filter(|i| {
                        i.auto_scaling_group_name.as_deref()
                            == Some(self.auto_scaling_group_name.as_str())
                    })
                    .map(|i| i.instance_id.unwrap_or_default())
                    .filter(|id| !id.is_empty())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Failed to get list of auto scaling
instances", &err))
    }

    pub async fn scale_min_size(&self, size: i32) -> Result<(), ScenarioError> {
```

```
        let update_group = self
            .autoscaling
            .update_auto_scaling_group()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .min_size(size)
            .send()
            .await;
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                format!("Failer to update group to min size ({size}))").as_str(),
                &err,
            ));
        }
        Ok(())
    }

    pub async fn scale_max_size(&self, size: i32) -> Result<(), ScenarioError> {
        // 5. UpdateAutoScalingGroup: update max size to 3.
        let update_group = self
            .autoscaling
            .update_auto_scaling_group()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .max_size(size)
            .send()
            .await;
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                format!("Failed to update group to max size ({size})").as_str(),
                &err,
            ));
        }
        Ok(())
    }

    pub async fn scale_desired_capacity(&self, capacity: i32) -> Result<(),
ScenarioError> {
        // 7. SetDesiredCapacity: set desired capacity to 2.
        //   Wait for a second instance to launch.
        let update_group = self
            .autoscaling
            .set_desired_capacity()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .desired_capacity(capacity)
            .send()
```

```
            .await;
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                format!("Failed to update group to desired capacity
({capacity}))").as_str(),
                &err,
            ));
        }
        Ok(())
    }

    pub async fn scale_group_to_zero(&self) -> Result<(), ScenarioError> {
        // If this fails it's fine, just means there are extra cloudwatch metrics
events for the scale-down.
        let _ = self
            .autoscaling
            .disable_metrics_collection()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .send()
            .await;

        // 12. DeleteAutoScalingGroup (to delete the group you must stop all
instances):
        //   UpdateAutoScalingGroup with MinSize=0
        let update_group = self
            .autoscaling
            .update_auto_scaling_group()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .min_size(0)
            .desired_capacity(0)
            .send()
            .await;
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                "Failed to update group for scaling down&",
                &err,
            ));
        }

        let stable = self.wait_for_stable(0).await;
        if let Err(err) = stable {
            return Err(ScenarioError::with(format!(
                "Error while waiting for group to be stable on scale down: {err}"
            )));
```

```
        }

        Ok(())
    }

    pub async fn terminate_some_instance(&self) -> Result<(), ScenarioError> {
        // Retrieve a list of instances in the auto scaling group.
        let auto_scaling_group = self.get_group().await?;
        let instances = auto_scaling_group.instances();
        // Or use other logic to find an instance to terminate.
        let instance = instances.first();
        if let Some(instance) = instance {
            let instance_id = if let Some(instance_id) = instance.instance_id() {
                instance_id
            } else {
                return Err(ScenarioError::with("Missing instance id"));
            };
            let termination = self
                .ec2
                .terminate_instances()
                .instance_ids(instance_id)
                .send()
                .await;
            if let Err(err) = termination {
                Err(ScenarioError::new(
                    "There was a problem terminating an instance",
                    &err,
                ))
            } else {
                Ok(())
            }
        } else {
            Err(ScenarioError::with("There was no instance to terminate"))
        }
    }
}

fn count_group_instances(group: &AutoScalingGroup) -> usize {
    group.instances.as_ref().map(|i| i.len()).unwrap_or(0)
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的以下主题。

- CreateAutoScalingGroup
- DeleteAutoScalingGroup
- DescribeAutoScalingGroups
- DescribeAutoScalingInstances
- DescribeScalingActivities
- DisableMetricsCollection
- EnableMetricsCollection
- SetDesiredCapacity
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

操作

`CreateAutoScalingGroup`

以下代码示例演示了如何使用 `CreateAutoScalingGroup`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CreateAutoScalingGroup于 Rust 的AWS SDK API 参考。

DeleteAutoScalingGroup

以下代码示例演示了如何使用 DeleteAutoScalingGroup。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteAutoScalingGroup于 Rust 的AWS SDK API 参考。

DescribeAutoScalingGroups

以下代码示例演示了如何使用 DescribeAutoScalingGroups。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeAutoScalingGroups于 Rust 的AWS SDK API 参考。

`DescribeAutoScalingInstances`

以下代码示例演示了如何使用 `DescribeAutoScalingInstances`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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))
   }
```

- 有关 API 的详细信息，请参阅适用DescribeAutoScalingInstances于 Rust 的AWS SDK API 参考。

DescribeScalingActivities

以下代码示例演示了如何使用 DescribeScalingActivities。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    pub async fn describe_scenario(&self) -> AutoScalingScenarioDescription {
        let group = self
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await
            .map(|s| {
                s.auto_scaling_groups()
                    .iter()
                    .map(|s| {
                        format!(
                            "{}: {}",
                            s.auto_scaling_group_name().unwrap_or("Unknown"),
                            s.status().unwrap_or("Unknown")
                        )
                    })
                    .collect::<Vec<String>>()
            })
            .map_err(|e| {
                ScenarioError::new("Failed to describe auto scaling groups for
scenario", &e)
            });

        let instances = self
            .list_instances()
            .await
            .map_err(|e| anyhow!("There was an error listing instances: {e}",));

        // 10. DescribeScalingActivities: list the scaling activities that have
occurred for the group so far.
```

```
        //   Bonus: use CloudWatch API to get and show some metrics collected for
the group.
        //   CW.ListMetrics with Namespace='AWS/AutoScaling' and
Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': }]
        //   CW.GetMetricStatistics with Statistics='Sum'. Start and End times must
be in UTC!
        let activities = self
            .autoscaling
            .describe_scaling_activities()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .into_paginator()
            .items()
            .send()
            .collect::<Result<Vec<_>, _>>()
            .await
            .map_err(|e| {
                anyhow!(
                    "There was an error retrieving scaling activities: {}",
                    DisplayErrorContext(&e)
                )
            });

        AutoScalingScenarioDescription {
            group,
            instances,
            activities,
        }
    }
```

- 有关 API 的详细信息，请参阅适用DescribeScalingActivities于 Rust 的AWS SDK API 参考。

`DisableMetricsCollection`

以下代码示例演示了如何使用 `DisableMetricsCollection`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        // 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;
```

- 有关 API 的详细信息，请参阅适用DisableMetricsCollection于 Rust 的AWS SDK API 参考。

`EnableMetricsCollection`

以下代码示例演示了如何使用 `EnableMetricsCollection`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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;
```

- 有关 API 的详细信息，请参阅适用EnableMetricsCollection于 Rust 的AWS SDK API 参考。

SetDesiredCapacity

以下代码示例演示了如何使用 SetDesiredCapacity。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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(())
   }
```

- 有关 API 的详细信息，请参阅适用SetDesiredCapacity于 Rust 的AWS SDK API 参考。

TerminateInstanceInAutoScalingGroup

以下代码示例演示了如何使用 TerminateInstanceInAutoScalingGroup。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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())
    }
```

- 有关 API 的详细信息，请参阅适用TerminateInstanceInAutoScalingGroup于 Rust 的AWS SDK API 参考。

UpdateAutoScalingGroup

以下代码示例演示了如何使用 UpdateAutoScalingGroup。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用UpdateAutoScalingGroup于 Rust 的AWS SDK API 参考。

使用适用于 Rust 的 SDK 的 Amazon Bedrock 运行时示例

以下代码示例向您展示了如何使用 AWS 适用于 Rust 的软件开发工具包和 Amazon Bedrock Runtime 来执行操作和实现常见场景。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 场景
- Anthropic Claude

场景

将工具与 Converse API 结合使用

以下代码示例展示了如何在应用程序、生成式 AI 模型和互联工具之间建立典型的交互，或者 APIs 如何调解 AI 与外界之间的交互。该代码示例以将外部天气 API 连接到人工智能模型模型为例，它可以根据用户输入提供实时天气信息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

演示的主要场景和逻辑。该脚本编排了用户、Amazon Bedrock Converse API 和天气工具之间的对话。

```
#[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();
}
```

演示使用的天气工具。该脚本定义了工具规范，并实现了从 Open-Meteo API 中检索天气数据的逻辑。

```
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()?)
}
```

用于打印消息内容块的实用程序。

```
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:?})"
        )))
    }
}
```

使用语句、错误实用程序和常量。

```
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(),
        })
```

```
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Converse。

Anthropic Claude

Converse

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Anthropic Claude 发送文本消息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 Bedrock 的 Converse API 向 Anthropic Claude 发送文本消息。

```
#[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)
}
```

使用语句、错误实用程序和常量。

```
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",
        })
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Converse。

ConverseStream

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Anthropic Claude 发送文本消息并实时处理响应流。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 Bedrock 的 API 向 Anthropic Claude 发送短信并直播回复令牌。 ConverseStream

```
#[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(),
    })
}
```

使用语句、错误实用程序和常量。

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

- 有关 API 的详细信息，请参阅适用ConverseStream于 Rust 的AWS SDK API 参考。

场景：将工具与 Converse API 搭配使用

以下代码示例展示了如何在应用程序、生成式 AI 模型和互联工具之间建立典型的交互，或者 APIs 如何调解 AI 与外界之间的交互。该代码示例以将外部天气 API 连接到人工智能模型模型为例，它可以根据用户输入提供实时天气信息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

演示的主要场景和逻辑。该脚本编排了用户、Amazon Bedrock Converse API 和天气工具之间的对话。

```
#[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();
}
```

演示使用的天气工具。该脚本定义了工具规范，并实现了从 Open-Meteo API 中检索天气数据的逻辑。

```
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()?)
}
```

用于打印消息内容块的实用程序。

```
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:?})"
        )))
    }
}
```

使用语句、错误实用程序和常量。

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

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Converse。

使用适用于 Rust 的 SDK 的 Amazon Bedrock 代理运行时示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS 软件开发工具包和 Amazon Bedrock Agents 运行时来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`InvokeAgent`

以下代码示例演示了如何使用 InvokeAgent。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
use aws_config::{BehaviorVersion, SdkConfig};
use aws_sdk_bedrockagentruntime::{
    self as bedrockagentruntime,
    types::{error::ResponseStreamError, ResponseStream},
};
#[allow(unused_imports)]
use mockall::automock;

const BEDROCK_AGENT_ID: &str = "AJBHXXILZN";
const BEDROCK_AGENT_ALIAS_ID: &str = "AVKP1ITZAA";
const BEDROCK_AGENT_REGION: &str = "us-east-1";

#[cfg(not(test))]
pub use EventReceiverImpl as EventReceiver;
#[cfg(test)]
pub use MockEventReceiverImpl as EventReceiver;

pub struct EventReceiverImpl {
    inner: aws_sdk_bedrockagentruntime::primitives::event_stream::EventReceiver<
        ResponseStream,
        ResponseStreamError,
    >,
}

#[cfg_attr(test, automock)]
impl EventReceiverImpl {
    #[allow(dead_code)]
```

```
    pub fn new(
        inner: aws_sdk_bedrockagentruntime::primitives::event_stream::EventReceiver<
            ResponseStream,
            ResponseStreamError,
        >,
    ) -> Self {
        Self { inner }
    }

    pub async fn recv(
        &mut self,
    ) -> Result<
        Option<ResponseStream>,
        aws_sdk_bedrockagentruntime::error::SdkError<
            ResponseStreamError,
            aws_smithy_types::event_stream::RawMessage,
        >,
    > {
        self.inner.recv().await
    }
}

#[tokio::main]
async fn main() -> Result<(), Box<bedrockagentruntime::Error>> {
    let result = invoke_bedrock_agent("I need help.".to_string(),
 "123".to_string()).await?;
    println!("{}", result);
    Ok(())
}

async fn invoke_bedrock_agent(
    prompt: String,
    session_id: String,
) -> Result<String, bedrockagentruntime::Error> {
    let sdk_config: SdkConfig = aws_config::defaults(BehaviorVersion::latest())
        .region(BEDROCK_AGENT_REGION)
        .load()
        .await;
    let bedrock_client = bedrockagentruntime::Client::new(&sdk_config);

    let command_builder = bedrock_client
        .invoke_agent()
        .agent_id(BEDROCK_AGENT_ID)
        .agent_alias_id(BEDROCK_AGENT_ALIAS_ID)
```

```
        .session_id(session_id)
        .input_text(prompt);

    let response = command_builder.send().await?;

    let response_stream = response.completion;

    let event_receiver = EventReceiver::new(response_stream);

    process_agent_response_stream(event_receiver).await
}

async fn process_agent_response_stream(
    mut event_receiver: EventReceiver,
) -> Result<String, bedrockagentruntime::Error> {
    let mut full_agent_text_response = String::new();

    while let Some(event_result) = event_receiver.recv().await? {
        match event_result {
            ResponseStream::Chunk(chunk) => {
                if let Some(bytes) = chunk.bytes {
                    match String::from_utf8(bytes.into_inner()) {
                        Ok(text_chunk) => {
                            full_agent_text_response.push_str(&text_chunk);
                        }
                        Err(e) => {
                            eprintln!("UTF-8 decoding error for chunk: {}", e);
                        }
                    }
                }
            }
            _ => {
                panic!("received an unhandled event type from Bedrock stream",);
            }
        }
    }
    Ok(full_agent_text_response)
}

#[cfg(test)]
mod test {

    use super::*;
```

```
    #[tokio::test]
    async fn test_process_agent_response_stream() {
        let mut mock = MockEventReceiverImpl::default();
        mock.expect_recv().times(1).returning(|| {
            Ok(Some(
                aws_sdk_bedrockagentruntime::types::ResponseStream::Chunk(
                    aws_sdk_bedrockagentruntime::types::PayloadPart::builder()
                        .set_bytes(Some(aws_smithy_types::Blob::new(vec![
                            116, 101, 115, 116, 32, 99, 111, 109, 112, 108, 101,
 116, 105, 111, 110,
                        ])))
                        .build(),
                ),
            ))
        });

        // end the stream
        mock.expect_recv().times(1).returning(|| Ok(None));

        let response = process_agent_response_stream(mock).await.unwrap();

        assert_eq!("test completion", response);
    }

    #[tokio::test]
    #[should_panic(expected = "received an unhandled event type from Bedrock
 stream")]
    async fn test_process_agent_response_stream_error() {
        let mut mock = MockEventReceiverImpl::default();
        mock.expect_recv().times(1).returning(|| {
            Ok(Some(
                aws_sdk_bedrockagentruntime::types::ResponseStream::Trace(

 aws_sdk_bedrockagentruntime::types::TracePart::builder().build(),
                ),
            ))
        });

        let _ = process_agent_response_stream(mock).await.unwrap();
    }
}
```

- 有关 API 的详细信息，请参阅适用InvokeAgent于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Amazon Cognito 身份提供者的代码示例

以下代码示例向您展示了如何使用带有 Amazon Cognito 身份提供商的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`ListUserPools`

以下代码示例演示了如何使用 `ListUserPools`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListUserPools于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Amazon Cognito Sync 示例

以下代码示例向您展示了如何使用带有 Amazon Cognito Sync 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`ListIdentityPoolUsage`

以下代码示例演示了如何使用 `ListIdentityPoolUsage`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListIdentityPoolUsage于 Rust 的AWS SDK API 参考。

使用适用于 Rust 的 SDK 的 Firehose 示例

以下代码示例向您展示了如何使用带有 Firehose 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`PutRecordBatch`

以下代码示例演示了如何使用 `PutRecordBatch`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
}
```

- 有关 API 的详细信息，请参阅适用PutRecordBatch于 Rust 的AWS SDK API 参考。

使用适用于 Rust 的 SDK 的 Amazon DocumentDB 示例

以下代码示例向您展示了如何使用适用于 Rust 的软件开发工具包和 Amazon Document AWS DB 来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 无服务器示例

无服务器示例

通过 Amazon DocumentDB 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 DocumentDB 更改流的记录而触发的事件。该函数检索 DocumentDB 有效负载，并记录下记录内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Rust 将 Amazon DocumentDB 事件与 Lambda 结合使用。

```
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(())

}
```

使用 SDK for Rust 的 DynamoDB 示例

以下代码示例向您展示了如何使用带有 DynamoDB 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务服务结合来完成特定任务的代码示例。

AWS 社区贡献就是由多个团队创建和维护的示例 AWS。要提供反馈，请使用链接存储库中提供的机制。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 场景
- 无服务器示例
- AWS 社区捐款

操作

`CreateTable`

以下代码示例演示了如何使用 `CreateTable`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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 create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .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))
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用CreateTable于 Rust 的AWS SDK API 参考。

`DeleteItem`

以下代码示例演示了如何使用 `DeleteItem`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)),
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteItem于 Rust 的AWS SDK API 参考。

`DeleteTable`

以下代码示例演示了如何使用 `DeleteTable`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DeleteTable于 Rust 的AWS SDK API 参考。

`ListTables`

以下代码示例演示了如何使用 `ListTables`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect::<Result<Vec<_>, _>>().await?;

    println!("Tables:");

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

    println!("Found {} tables", table_names.len());
    Ok(table_names)
}
```

确定表是否存在。

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

- 有关 API 的详细信息，请参阅适用ListTables于 Rust 的AWS SDK API 参考。

`PutItem`

以下代码示例演示了如何使用 `PutItem`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
```

```
        username, first_name, last_name, age, p_type
    );

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

- 有关 API 的详细信息，请参阅适用PutItem于 Rust 的AWS SDK API 参考。

Query

以下代码示例演示了如何使用 Query。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

查找指定年份制作的电影。

```
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![])
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Query。

Scan

以下代码示例演示了如何使用 Scan。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Scan。

场景

连接到本地实例

以下代码示例演示如何覆盖终端节点 URL 以连接到 DynamoDB 和软件开发工具包的本地开发部署。AWS

有关更多信息，请参阅 DynamoDB Local。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 Rust 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

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

使用 PartiQL 来查询表

以下代码示例展示了如何：

- 通过运行 SELECT 语句来获取项目。

- 通过运行 INSERT 语句来添加项目。
- 通过运行 UPDATE 语句来更新项目。
- 通过运行 DELETE 语句来删除项目。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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");

    match client
        .create_table()
        .table_name(table)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .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 {{
                "{}": ?,
                "acount_type": ?,
                "age": ?,
                "first_name": ?,
                "last_name": ?
        }} "#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![
            AttributeValue::S(item.utype),
            AttributeValue::S(item.age),
            AttributeValue::S(item.first_name),
            AttributeValue::S(item.last_name),
        ]))
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
```

```
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {
            println!("Got an error querying table:");
            println!("{}", e);
            process::exit(1);
        }
    }
}

async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
 Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;

    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用ExecuteStatement于 Rust 的AWS SDK API 参考。

保存 EXIF 和其他图像信息

以下代码示例展示了如何：

- 从 JPG、JPEG 或 PNG 文件中获取 EXIF 信息。

- 将图像文件上传到 Amazon S3 存储桶。
- 使用 Amazon Rekognition 识别文件中的三个主要属性（标签）。
- 将 EXIF 和标签信息添加到该区域的 Amazon DynamoDB 表中。

适用于 Rust 的 SDK

从 JPG、JPEG 或 PNG 文件中获取 EXIF 信息，将图像文件上传到 Amazon S3 存储桶，使用 Amazon Rekognition 识别文件中的三个主要属性（Amazon Rekognition 中的标签），然后将 EXIF 和标签信息添加到该区域的 Amazon DynamoDB 表中。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Amazon Rekognition
- Amazon S3

无服务器示例

通过 DynamoDB 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 DynamoDB 流的记录而触发的事件。该函数检索 DynamoDB 有效负载，并记录下记录内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Rust 将 DynamoDB 事件与 Lambda 结合使用。

```
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(())

}
```

通过 DynamoDB 触发器报告 Lambda 函数批处理项目失败

以下代码示例演示如何为接收来自 DynamoDB 流的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Rust 通过 Lambda 进行 DynamoDB 批处理项目失败。

```
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 社区捐款

构建和测试无服务器应用程序

以下代码示例演示如何使用 API Gateway 与 Lambda 和 DynamoDB 构建和测试无服务器应用程序

适用于 Rust 的 SDK

演示如何使用 Rust SDK 构建和测试包含 API Gateway 以及 Lambda 和 DynamoDB 的无服务器应用程序。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda

使用 SDK for Rust 的 Amazon EBS 示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS 软件开发工具包和 Amazon EBS 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

CompleteSnapshot

以下代码示例演示了如何使用 CompleteSnapshot。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CompleteSnapshot于 Rust 的AWS SDK API 参考。

PutSnapshotBlock

以下代码示例演示了如何使用 PutSnapshotBlock。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用PutSnapshotBlock于 Rust 的AWS SDK API 参考。

`StartSnapshot`

以下代码示例演示了如何使用 `StartSnapshot`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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())
}
```

- 有关 API 的详细信息，请参阅适用StartSnapshot于 Rust 的AWS SDK API 参考。

使用适用于 Rust 的软件开发工具包的亚马逊 EC2 示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS 软件开发工具包和 Amazon 来执行操作和实现常见场景 EC2。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能

- 操作

开始使用

你好 Amazon EC2

以下代码示例显示了如何开始使用 Amazon EC2。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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}");
```

```
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用DescribeSecurityGroups于 Rust 的AWS SDK API 参考。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建密钥对和安全组。
- 选择 Amazon 机器映像 (AMI) 和兼容的实例类型，然后创建实例。
- 停止实例，然后再重启。
- 将弹性 IP 地址与您的实例相关联。
- 使用 SSH 连接到您的实例，然后清理资源。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

该 EC2InstanceScenario 实现包含将示例作为一个整体运行的逻辑。

```
//! 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 结构用作测试的自动模组点，其函数封装了 SDK 调用。 EC2

```
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 结构体用作测试的自动模拟点，其函数封装了 SSM SDK 调用。

```
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)
    }
}
```

此场景使用多个“管理器”风格的结构体，来处理对在整个场景中创建和删除的资源的访问。

```
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."),
        }
    }
}
```

场景的主入口点。

```
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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的以下主题。
  - AllocateAddress
  - AssociateAddress
  - AuthorizeSecurityGroupIngress
  - CreateKeyPair
  - CreateSecurityGroup
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DescribeImages
  - DescribeInstanceTypes
  - DescribeInstances
  - DescribeKeyPairs
  - DescribeSecurityGroups
  - DisassociateAddress
  - ReleaseAddress
  - RunInstances
  - StartInstances
  - StopInstances
  - TerminateInstances
  - UnmonitorInstances

操作

`AllocateAddress`

以下代码示例演示了如何使用 `AllocateAddress`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用AllocateAddress于 Rust 的AWS SDK API 参考。

`AssociateAddress`

以下代码示例演示了如何使用 `AssociateAddress`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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)
    }
```

- 有关 API 的详细信息，请参阅适用AssociateAddress于 Rust 的AWS SDK API 参考。

`AuthorizeSecurityGroupIngress`

以下代码示例演示了如何使用 `AuthorizeSecurityGroupIngress`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用AuthorizeSecurityGroupIngress于 Rust 的AWS SDK API 参考。

`CreateKeyPair`

以下代码示例演示了如何使用 `CreateKeyPair`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Rust 实现，它调用 EC2 客户端的 create_key_pair 并提取返回的材料。

```
    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))
```

```
    }
```

一个函数，它调用 create_key impl 并安全地保存 PEM 私有密钥。

```
    /// 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)
    }
```

- 有关 API 的详细信息，请参阅适用CreateKeyPair于 Rust 的AWS SDK API 参考。

CreateSecurityGroup

以下代码示例演示了如何使用 CreateSecurityGroup。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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)
    }
```

- 有关 API 的详细信息，请参阅适用CreateSecurityGroup于 Rust 的AWS SDK API 参考。

`CreateTags`

以下代码示例演示了如何使用 `CreateTags`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例在创建实例后应用 Name 标签。

```
    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())
    }
```

- 有关 API 的详细信息，请参阅适用CreateTags于 Rust 的AWS SDK API 参考。

DeleteKeyPair

以下代码示例演示了如何使用 DeleteKeyPair。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

围绕 delete_key 的包装器，它也会移除备份的私有 PEM 密钥。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteKeyPair于 Rust 的AWS SDK API 参考。

`DeleteSecurityGroup`

以下代码示例演示了如何使用 `DeleteSecurityGroup`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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(())
    }
```

- 有关 API 的详细信息，请参阅适用DeleteSecurityGroup于 Rust 的AWS SDK API 参考。

`DeleteSnapshot`

以下代码示例演示了如何使用 `DeleteSnapshot`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

    println!("Deleted");

    Ok(())
```

```
}
```

- 有关 API 的详细信息，请参阅适用DeleteSnapshot于 Rust 的AWS SDK API 参考。

`DescribeImages`

以下代码示例演示了如何使用 `DescribeImages`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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)
       }
   }
```

将 list_images 函数与 SSM 结合使用，以根据环境进行限制。有关 SSM 的更多详细信息，请参阅 https://docs.aws.amazon.com/systems-manager/ latest/userguide/example _ss GetParameters m_ _section.html。

```
    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)
    }
```

- 有关 API 的详细信息，请参阅适用DescribeImages于 Rust 的AWS SDK API 参考。

`DescribeInstanceStatus`

以下代码示例演示了如何使用 `DescribeInstanceStatus`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeInstanceStatus于 Rust 的AWS SDK API 参考。

`DescribeInstanceTypes`

以下代码示例演示了如何使用 `DescribeInstanceTypes`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DescribeInstanceTypes于 Rust 的AWS SDK API 参考。

DescribeInstances

以下代码示例演示了如何使用 DescribeInstances。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

检索 EC2 实例的详细信息。

```
   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())
   }
```

创建 EC2 实例后，检索并存储其详细信息。

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

- 有关 API 的详细信息，请参阅适用DescribeInstances于 Rust 的AWS SDK API 参考。

DescribeKeyPairs

以下代码示例演示了如何使用 DescribeKeyPairs。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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())
}
```

- 有关 API 的详细信息，请参阅适用DescribeKeyPairs于 Rust 的AWS SDK API 参考。

DescribeRegions

以下代码示例演示了如何使用 DescribeRegions。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeRegions于 Rust 的AWS SDK API 参考。

`DescribeSecurityGroups`

以下代码示例演示了如何使用 `DescribeSecurityGroups`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DescribeSecurityGroups于 Rust 的AWS SDK API 参考。

`DescribeSnapshots`

以下代码示例演示了如何使用 `DescribeSnapshots`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

显示快照的状态。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeSnapshots于 Rust 的AWS SDK API 参考。

`DisassociateAddress`

以下代码示例演示了如何使用 `DisassociateAddress`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DisassociateAddress于 Rust 的AWS SDK API 参考。

`RebootInstances`

以下代码示例演示了如何使用 `RebootInstances`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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(())
    }
```

使用等待器 API 等待实例进入已停止和就绪状态。使用等待器 API 要求在 rust 文件中添加“use aws_sdk_ec2::client::Waiter”。

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

- 有关 API 的详细信息，请参阅适用RebootInstances于 Rust 的AWS SDK API 参考。

`ReleaseAddress`

以下代码示例演示了如何使用 `ReleaseAddress`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    pub async fn deallocate_ip_address(&self, allocation_id: &str) -> Result<(),
EC2Error> {
```

```
        self.client
            .release_address()
            .allocation_id(allocation_id)
            .send()
            .await?;
        Ok(())
    }
```

- 有关 API 的详细信息，请参阅适用ReleaseAddress于 Rust 的AWS SDK API 参考。

`RunInstances`

以下代码示例演示了如何使用 `RunInstances`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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())
    }
```

- 有关 API 的详细信息，请参阅适用RunInstances于 Rust 的AWS SDK API 参考。

`StartInstances`

以下代码示例演示了如何使用 `StartInstances`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

按 EC2 实例 ID 启动实例。

```
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(())
}
```

使用等待器 API 等待实例进入就绪且状态正常的状态。使用等待器 API 要求在 rust 文件中添加“use aws_sdk_ec2::client::Waiter”。

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

- 有关 API 的详细信息，请参阅适用StartInstances于 Rust 的AWS SDK API 参考。

`StopInstances`

以下代码示例演示了如何使用 `StopInstances`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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(())
```

```
    }
```

使用等待器 API 等待实例进入已停止状态。使用等待器 API 要求在 rust 文件中添加“use aws_sdk_ec2::client::Waiter”。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用StopInstances于 Rust 的AWS SDK API 参考。

`TerminateInstances`

以下代码示例演示了如何使用 `TerminateInstances`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
    }
```

使用等待器 API 等待实例进入已终止状态。使用等待器 API 要求在 rust 文件中添加“use aws_sdk_ec2::client::Waiter”。

```
    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(())
    }
```

- 有关 API 的详细信息，请参阅适用TerminateInstances于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Amazon ECR 示例

以下代码示例向您展示了如何使用带有 Amazon ECR 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`DescribeRepositories`

以下代码示例演示了如何使用 `DescribeRepositories`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeRepositories于 Rust 的AWS SDK API 参考。

`ListImages`

以下代码示例演示了如何使用 `ListImages`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListImages于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Amazon ECS 示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS 软件开发工具包和 Amazon ECS 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`CreateCluster`

以下代码示例演示了如何使用 `CreateCluster`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CreateCluster于 Rust 的AWS SDK API 参考。

DeleteCluster

以下代码示例演示了如何使用 DeleteCluster。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteCluster于 Rust 的AWS SDK API 参考。

DescribeClusters

以下代码示例演示了如何使用 DescribeClusters。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeClusters于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Amazon EKS 示例

以下代码示例向您展示了如何使用适用于 Rust 的软件开发工具包和 Amazon E AWS KS 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`CreateCluster`

以下代码示例演示了如何使用 `CreateCluster`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CreateCluster于 Rust 的AWS SDK API 参考。

`DeleteCluster`

以下代码示例演示了如何使用 `DeleteCluster`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteCluster于 Rust 的AWS SDK API 参考。

AWS Glue 使用 Rust 版 SDK 的示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS SDK 来执行操作和实现常见场景 AWS Glue。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

你好 AWS Glue

以下代码示例展示了如何开始使用 AWS Glue。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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)),
            }
        }
```

- 有关 API 的详细信息，请参阅适用ListJobs于 Rust 的AWS SDK API 参考。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建爬网程序，爬取公有 Amazon S3 存储桶并生成包含 CSV 格式的元数据的数据库。
- 列出您的中的数据库和表的相关信息 AWS Glue Data Catalog。
- 创建任务，从 S3 存储桶提取 CSV 数据，转换数据，然后将 JSON 格式的输出加载到另一个 S3 存储桶中。
- 列出有关作业运行的信息，查看转换后的数据，并清除资源。

有关更多信息，请参阅教程： AWS Glue Studio 入门。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建并运行爬网程序，爬取公共 Amazon Simple Storage Service（Amazon S3）存储桶并生成一个描述其找到的 CSV 格式数据的元数据数据库。

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

列出您的中的数据库和表的相关信息 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();
```

创建并运行任务，从源 Amazon S3 存储桶提取 CSV 数据，通过删除和重命名字段对其进行转换，然后将 JSON 格式的输出加载到另一个 Amazon S3 存储桶中。

```
        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();
```

删除演示创建的所有资源。

```
        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)?;
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun

- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

操作

`CreateCrawler`

以下代码示例演示了如何使用 `CreateCrawler`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用CreateCrawler于 Rust 的AWS SDK API 参考。

`CreateJob`

以下代码示例演示了如何使用 `CreateJob`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用CreateJob于 Rust 的AWS SDK API 参考。

`DeleteCrawler`

以下代码示例演示了如何使用 `DeleteCrawler`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DeleteCrawler于 Rust 的AWS SDK API 参考。

`DeleteDatabase`

以下代码示例演示了如何使用 `DeleteDatabase`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        glue.delete_database()
```

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

- 有关 API 的详细信息，请参阅适用DeleteDatabase于 Rust 的AWS SDK API 参考。

`DeleteJob`

以下代码示例演示了如何使用 `DeleteJob`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DeleteJob于 Rust 的AWS SDK API 参考。

`DeleteTable`

以下代码示例演示了如何使用 `DeleteTable`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DeleteTable于 Rust 的AWS SDK API 参考。

`GetCrawler`

以下代码示例演示了如何使用 `GetCrawler`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
            let tmp_crawler = glue
                .get_crawler()
                .name(self.crawler())
                .send()
                .await
                .map_err(GlueMvpError::from_glue_sdk)?;
```

- 有关 API 的详细信息，请参阅适用GetCrawler于 Rust 的AWS SDK API 参考。

`GetDatabase`

以下代码示例演示了如何使用 `GetDatabase`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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()))?;
```

- 有关 API 的详细信息，请参阅适用GetDatabase于 Rust 的AWS SDK API 参考。

`GetJobRun`

以下代码示例演示了如何使用 `GetJobRun`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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();
        }
```

- 有关 API 的详细信息，请参阅适用GetJobRun于 Rust 的AWS SDK API 参考。

`GetTables`

以下代码示例演示了如何使用 `GetTables`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        let tables = glue
```

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

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

- 有关 API 的详细信息，请参阅适用GetTables于 Rust 的AWS SDK API 参考。

`ListJobs`

以下代码示例演示了如何使用 `ListJobs`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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)),
            }
        }
```

- 有关 API 的详细信息，请参阅适用ListJobs于 Rust 的AWS SDK API 参考。

`StartCrawler`

以下代码示例演示了如何使用 `StartCrawler`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用StartCrawler于 Rust 的AWS SDK API 参考。

`StartJobRun`

以下代码示例演示了如何使用 `StartJobRun`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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();
```

- 有关 API 的详细信息，请参阅适用StartJobRun于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 IAM 示例

以下代码示例向您展示了如何使用带有 IAM 的 Rust AWS 开发工具包来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

开始使用 IAM

以下代码示例展示了如何开始使用 IAM。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

来自 src/bin/hello .rs.

```
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(())
}
```

来自 src/ .r iam-service-lib s.

```
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)
}
```

- 有关 API 的详细信息，请参阅适用ListPolicies于 Rust 的AWS SDK API 参考。

基本功能

了解基本功能

以下代码示例展示了如何创建用户并代入角色。

> **Warning**
>
> 为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供者的联合身份验证，例如 AWS IAM Identity Center。

- 创建没有权限的用户。
- 创建授予列出账户的 Amazon S3 存储桶的权限的角色
- 添加策略以允许用户代入该角色。
- 代入角色并使用临时凭证列出 S3 存储桶，然后清除资源。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的以下主题。
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

操作

`AttachRolePolicy`

以下代码示例演示了如何使用 `AttachRolePolicy`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
}
```

- 有关 API 的详细信息，请参阅适用AttachRolePolicy于 Rust 的AWS SDK API 参考。

`AttachUserPolicy`

以下代码示例演示了如何使用 `AttachUserPolicy`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用AttachUserPolicy于 Rust 的AWS SDK API 参考。

`CreateAccessKey`

以下代码示例演示了如何使用 `CreateAccessKey`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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())
}
```

- 有关 API 的详细信息，请参阅适用CreateAccessKey于 Rust 的AWS SDK API 参考。

`CreatePolicy`

以下代码示例演示了如何使用 `CreatePolicy`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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())
}
```

- 有关 API 的详细信息，请参阅适用CreatePolicy于 Rust 的AWS SDK API 参考。

`CreateRole`

以下代码示例演示了如何使用 `CreateRole`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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())
}
```

- 有关 API 的详细信息，请参阅适用CreateRole于 Rust 的AWS SDK API 参考。

CreateServiceLinkedRole

以下代码示例演示了如何使用 CreateServiceLinkedRole。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用CreateServiceLinkedRole于 Rust 的AWS SDK API 参考。

CreateUser

以下代码示例演示了如何使用 CreateUser。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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())
}
```

- 有关 API 的详细信息，请参阅适用CreateUser于 Rust 的AWS SDK API 参考。

`DeleteAccessKey`

以下代码示例演示了如何使用 `DeleteAccessKey`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteAccessKey于 Rust 的AWS SDK API 参考。

`DeletePolicy`

以下代码示例演示了如何使用 `DeletePolicy`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
pub async fn delete_policy(client: &iamClient, policy: Policy) -> Result<(),
 iamError> {
    client
        .delete_policy()
        .policy_arn(policy.arn.unwrap())
        .send()
        .await?;
    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用DeletePolicy于 Rust 的AWS SDK API 参考。

DeleteRole

以下代码示例演示了如何使用 DeleteRole。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteRole于 Rust 的AWS SDK API 参考。

DeleteServiceLinkedRole

以下代码示例演示了如何使用 DeleteServiceLinkedRole。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteServiceLinkedRole于 Rust 的AWS SDK API 参考。

`DeleteUser`

以下代码示例演示了如何使用 `DeleteUser`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
}
```

- 有关 API 的详细信息，请参阅适用DeleteUser于 Rust 的AWS SDK API 参考。

`DeleteUserPolicy`

以下代码示例演示了如何使用 `DeleteUserPolicy`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteUserPolicy于 Rust 的AWS SDK API 参考。

`DetachRolePolicy`

以下代码示例演示了如何使用 `DetachRolePolicy`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DetachRolePolicy于 Rust 的AWS SDK API 参考。

DetachUserPolicy

以下代码示例演示了如何使用 DetachUserPolicy。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DetachUserPolicy于 Rust 的AWS SDK API 参考。

GetAccountPasswordPolicy

以下代码示例演示了如何使用 GetAccountPasswordPolicy。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
pub async fn get_account_password_policy(
    client: &iamClient,
) -> Result<GetAccountPasswordPolicyOutput, SdkError<GetAccountPasswordPolicyError>>
 {
    let response = client.get_account_password_policy().send().await?;

    Ok(response)
}
```

- 有关 API 的详细信息，请参阅适用GetAccountPasswordPolicy于 Rust 的AWS SDK API 参考。

`GetRole`

以下代码示例演示了如何使用 `GetRole`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用GetRole于 Rust 的AWS SDK API 参考。

`ListAttachedRolePolicies`

以下代码示例演示了如何使用 `ListAttachedRolePolicies`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用ListAttachedRolePolicies于 Rust 的AWS SDK API 参考。

`ListGroups`

以下代码示例演示了如何使用 `ListGroups`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
pub async fn list_groups(
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListGroupsOutput, SdkError<ListGroupsError>> {
    let response = client
        .list_groups()
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- 有关 API 的详细信息，请参阅适用ListGroups于 Rust 的AWS SDK API 参考。

`ListPolicies`

以下代码示例演示了如何使用 `ListPolicies`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用ListPolicies于 Rust 的AWS SDK API 参考。

`ListRolePolicies`

以下代码示例演示了如何使用 `ListRolePolicies`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用ListRolePolicies于 Rust 的AWS SDK API 参考。

`ListRoles`

以下代码示例演示了如何使用 `ListRoles`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用ListRoles于 Rust 的AWS SDK API 参考。

`ListSAMLProviders`

以下代码示例演示了如何使用 `ListSAMLProviders`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
pub async fn list_saml_providers(
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}
```

- 有关 API 的详细信息，请参阅适用于 Rust 的AWS SDK SAMLProviders 中列出 API 参考。

`ListUsers`

以下代码示例演示了如何使用 `ListUsers`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用ListUsers于 Rust 的AWS SDK API 参考。

AWS IoT 使用 Rust 版 SDK 的示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS SDK 来执行操作和实现常见场景 AWS IoT。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`DescribeEndpoint`

以下代码示例演示了如何使用 `DescribeEndpoint`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeEndpoint于 Rust 的AWS SDK API 参考。

`ListThings`

以下代码示例演示了如何使用 `ListThings`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListThings于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Kinesis 示例

以下代码示例向您展示了如何使用带有 Kinesis 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 无服务器示例

操作

`CreateStream`

以下代码示例演示了如何使用 `CreateStream`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CreateStream于 Rust 的AWS SDK API 参考。

DeleteStream

以下代码示例演示了如何使用 DeleteStream。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用DeleteStream于 Rust 的AWS SDK API 参考。

DescribeStream

以下代码示例演示了如何使用 DescribeStream。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeStream于 Rust 的AWS SDK API 参考。

`ListStreams`

以下代码示例演示了如何使用 `ListStreams`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListStreams于 Rust 的AWS SDK API 参考。

PutRecord

以下代码示例演示了如何使用 PutRecord。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用PutRecord于 Rust 的AWS SDK API 参考。

无服务器示例

通过 Kinesis 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 Kinesis 流的记录而触发的事件。该函数检索 Kinesis 有效负载，将 Base64 解码，并记录下记录内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 Kinesis 事件与 Lambda 结合使用。

```
// 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
}
```

通过 Kinesis 触发器报告 Lambda 函数批处理项目失败

以下代码示例展示了如何为接收来自 Kinesis 流的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告通过 Rust 进行 Lambda Kinesis 批处理项目失败。

```
// 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 使用 Rust 版 SDK 的示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS SDK 来执行操作和实现常见场景 AWS KMS。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`CreateKey`

以下代码示例演示了如何使用 `CreateKey`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CreateKey于 Rust 的AWS SDK API 参考。

`Decrypt`

以下代码示例演示了如何使用 `Decrypt`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Decrypt。

`Encrypt`

以下代码示例演示了如何使用 `Encrypt`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Encrypt。

`GenerateDataKey`

以下代码示例演示了如何使用 `GenerateDataKey`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用GenerateDataKey于 Rust 的AWS SDK API 参考。

`GenerateDataKeyWithoutPlaintext`

以下代码示例演示了如何使用 `GenerateDataKeyWithoutPlaintext`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用GenerateDataKeyWithoutPlaintext于 Rust 的AWS SDK API 参考。

`GenerateRandom`

以下代码示例演示了如何使用 `GenerateRandom`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用GenerateRandom于 Rust 的AWS SDK API 参考。

`ListKeys`

以下代码示例演示了如何使用 `ListKeys`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListKeys于 Rust 的AWS SDK API 参考。

`ReEncrypt`

以下代码示例演示了如何使用 `ReEncrypt`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ReEncrypt于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Lambda 示例

以下代码示例向您展示了如何使用带有 Lambda 的 Rust AWS 开发工具包来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务服务结合来完成特定任务的代码示例。

AWS 社区贡献就是由多个团队创建和维护的示例 AWS。要提供反馈，请使用链接存储库中提供的机制。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 基本功能
- 操作

- 场景
- 无服务器示例
- AWS 社区捐款

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建 IAM 角色和 Lambda 函数，然后上传处理程序代码。
- 使用单个参数来调用函数并获取结果。
- 更新函数代码并使用环境变量进行配置。
- 使用新参数来调用函数并获取结果。显示返回的执行日志。
- 列出账户函数，然后清除函数。

有关更多信息，请参阅使用控制台创建 Lambda 函数。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在这种情况下使用的带依赖项的 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"
```

可就这种情况简化 Lambda 调用的一组实用程序。此文件在 crate 中是 src/ations.rs。

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

一个从头到尾运行场景的二进制文件，使用命令行标志来控制某些行为。这个文件是箱子里的 src/bin/scenario .rs。

```
/*
## 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");
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的以下主题。
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

操作

`CreateFunction`

以下代码示例演示了如何使用 `CreateFunction`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * 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())
    }
```

- 有关 API 的详细信息，请参阅适用CreateFunction于 Rust 的AWS SDK API 参考。

DeleteFunction

以下代码示例演示了如何使用 DeleteFunction。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   /** 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)
}
```

- 有关 API 的详细信息，请参阅适用DeleteFunction于 Rust 的AWS SDK API 参考。

GetFunction

以下代码示例演示了如何使用 GetFunction。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/** 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)
}
```

- 有关 API 的详细信息，请参阅适用GetFunction于 Rust 的AWS SDK API 参考。

Invoke

以下代码示例演示了如何使用 Invoke。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /** 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(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")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的调用。

`ListFunctions`

以下代码示例演示了如何使用 `ListFunctions`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /** 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)
    }
```

- 有关 API 的详细信息，请参阅适用ListFunctions于 Rust 的AWS SDK API 参考。

`UpdateFunctionCode`

以下代码示例演示了如何使用 `UpdateFunctionCode`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /** 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())
    }
```

- 有关 API 的详细信息，请参阅适用UpdateFunctionCode于 Rust 的AWS SDK API 参考。

UpdateFunctionConfiguration

以下代码示例演示了如何使用 UpdateFunctionConfiguration。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /** 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)
    }
```

- 有关 API 的详细信息，请参阅适用UpdateFunctionConfiguration于 Rust 的AWS SDK API 参考。

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 Rust 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

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

无服务器示例

使用 Lambda 函数连接到 Amazon RDS 数据库

以下代码示例显示如何实现连接到 RDS 数据库的 Lambda 函数。该函数发出一个简单的数据库请求并返回结果。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

在 Lambda 函数中使用 Rust 连接到 Amazon RDS 数据库。

```
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}")
    }))
}
```

通过 Kinesis 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 Kinesis 流的记录而触发的事件。该函数检索 Kinesis 有效负载，将 Base64 解码，并记录下记录内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 Kinesis 事件与 Lambda 结合使用。

```
// 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
}
```

通过 DynamoDB 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 DynamoDB 流的记录而触发的事件。该函数检索 DynamoDB 有效负载，并记录下记录内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Rust 将 DynamoDB 事件与 Lambda 结合使用。

```
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(())

}
```

通过 Amazon DocumentDB 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 DocumentDB 更改流的记录而触发的事件。该函数检索 DocumentDB 有效负载，并记录下记录内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Rust 将 Amazon DocumentDB 事件与 Lambda 结合使用。

```
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(())

}
```

通过 Amazon MSK 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 Amazon MSK 集群的记录而触发的事件。该函数检索 MSK 有效负载，并记录下记录内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 Amazon MSK 事件与 Lambda 结合使用。

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-
started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/awslabs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error> {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

         let record_text = record.value.as_ref().ok_or("Value is None")?;
         info!("Record: {}", &record_text);
```

```
            // perform Base64 decoding
            let record_bytes = BASE64_STANDARD.decode(record_text)?;
            let message = std::str::from_utf8(&record_bytes)?;

            info!("Message: {}", message);
           }

    }

    Ok(().into())
}

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

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```

通过 Amazon S3 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收通过将对象上传到 S3 桶而触发的事件。该函数从事件参数中检索 S3 存储桶名称和对象密钥，并调用 Amazon S3 API 来检索和记录对象的内容类型。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Rust 将 S3 事件与 Lambda 结合使用。

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

通过 Amazon SNS 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 主题的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 SNS 事件与 Lambda 结合使用。

```
// 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
}
```

通过 Amazon SQS 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 队列的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 SQS 事件与 Lambda 结合使用。

```
// 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
}
```

通过 Kinesis 触发器报告 Lambda 函数批处理项目失败

以下代码示例展示了如何为接收来自 Kinesis 流的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告通过 Rust 进行 Lambda Kinesis 批处理项目失败。

```
// 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
}
```

通过 DynamoDB 触发器报告 Lambda 函数批处理项目失败

以下代码示例演示如何为接收来自 DynamoDB 流的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Rust 通过 Lambda 进行 DynamoDB 批处理项目失败。

```
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
}
```

报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

以下代码示例展示了如何为接收来自 SQS 队列的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Rust 进行 Lambda SQS 批处理项目失败。

```
// 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 社区捐款

构建和测试无服务器应用程序

以下代码示例演示如何使用 API Gateway 与 Lambda 和 DynamoDB 构建和测试无服务器应用程序

适用于 Rust 的 SDK

演示如何使用 Rust SDK 构建和测试包含 API Gateway 以及 Lambda 和 DynamoDB 的无服务器应用程序。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda

MediaLive 使用 Rust 版 SDK 的示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS SDK 来执行操作和实现常见场景 MediaLive。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`ListInputs`

以下代码示例演示了如何使用 `ListInputs`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在“区域” ARNs 中列出您的 MediaLive 输入名称。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListInputs于 Rust 的AWS SDK API 参考。

MediaPackage 使用 Rust 版 SDK 的示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS SDK 来执行操作和实现常见场景 MediaPackage。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`ListChannels`

以下代码示例演示了如何使用 `ListChannels`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出频道 ARNs 和描述。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListChannels于 Rust 的AWS SDK API 参考。

`ListOriginEndpoints`

以下代码示例演示了如何使用 `ListOriginEndpoints`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出您的终端节点描述和 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(())
}
```

- 有关 API 的详细信息，请参阅适用ListOriginEndpoints于 Rust 的AWS SDK API 参考。

使用适用于 Rust 的 SDK 的 Amazon MSK 示例

以下代码示例向您展示了如何使用带有 Amazon M AWS SK 的 Rust 开发工具包来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 无服务器示例

无服务器示例

通过 Amazon MSK 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 Amazon MSK 集群的记录而触发的事件。该函数检索 MSK 有效负载，并记录下记录内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 Amazon MSK 事件与 Lambda 结合使用。

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-
started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/awslabs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error> {

    let payload = event.payload.records;
```

```
    for (_name, records) in payload.iter() {

        for record in records {

         let record_text = record.value.as_ref().ok_or("Value is None")?;
         info!("Record: {}", &record_text);

         // perform Base64 decoding
         let record_bytes = BASE64_STANDARD.decode(record_text)?;
         let message = std::str::from_utf8(&record_bytes)?;

         info!("Message: {}", message);
        }

    }

    Ok(().into())
}

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

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```

使用 SDK for Rust 的 Amazon Polly 示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS 软件开发工具包和 Amazon Polly 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 场景

操作

`DescribeVoices`

以下代码示例演示了如何使用 `DescribeVoices`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeVoices于 Rust 的AWS SDK API 参考。

`ListLexicons`

以下代码示例演示了如何使用 `ListLexicons`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListLexicons于 Rust 的AWS SDK API 参考。

PutLexicon

以下代码示例演示了如何使用 PutLexicon。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用PutLexicon于 Rust 的AWS SDK API 参考。

SynthesizeSpeech

以下代码示例演示了如何使用 SynthesizeSpeech。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用SynthesizeSpeech于 Rust 的AWS SDK API 参考。

场景

将文本转换为语音以及将语音转换回文本

以下代码示例展示了如何：

- 使用 Amazon Polly 将纯文本 (UTF-8) 输入文件合成为音频文件。
- 将音频文件上传到 Amazon S3 存储桶。
- 使用 Amazon Transcribe 将音频文件转换为文本。
- 显示文本。

适用于 Rust 的 SDK

使用 Amazon Polly 将纯文本（UTF-8）输入文件合成为音频文件，将音频文件上传到 Amazon S3 存储桶，使用 Amazon Transcribe 将该音频文件转换为文本，然后显示文本。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Polly
- Amazon S3
- Amazon Transcribe

使用适用于 Rust 的 SDK 的 Amazon RDS 示例

以下代码示例向您展示了如何使用适用于 Rust 的软件开发工具包和 Amazon R AWS DS 来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 无服务器示例

无服务器示例

使用 Lambda 函数连接到 Amazon RDS 数据库

以下代码示例显示如何实现连接到 RDS 数据库的 Lambda 函数。该函数发出一个简单的数据库请求并返回结果。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

在 Lambda 函数中使用 Rust 连接到 Amazon RDS 数据库。

```
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}")
    }))
}
```

使用 SDK for Rust 的 Amazon RDS 数据服务示例

以下代码示例向您展示了如何使用适用于 Rust 的软件开发工具包和 Amazon R AWS DS 数据服务来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`ExecuteStatement`

以下代码示例演示了如何使用 `ExecuteStatement`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ExecuteStatement于 Rust 的AWS SDK API 参考。

使用适用于 Rust 的 SDK 的 Amazon Rekognition 示例

以下代码示例向您展示了如何使用带有 Amazon Rekognition 的 Rust AWS 开发工具包来执行操作和实现常见场景。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 场景

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 Rust 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

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

检测图像中的人脸

以下代码示例展示了如何：

- 将图像保存到 Amazon S3 存储桶中。
- 使用 Amazon Rekognition 检测面部细节，例如年龄范围、性别和情绪（如微笑）。
- 显示这些细节。

适用于 Rust 的 SDK

将图像保存到具有 uploads 前缀的 Amazon S3 存储桶中，使用 Amazon Rekognition 检测面部细节，例如年龄范围、性别和情绪（微笑等），并显示这些细节。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3

保存 EXIF 和其他图像信息

以下代码示例展示了如何：

- 从 JPG、JPEG 或 PNG 文件中获取 EXIF 信息。
- 将图像文件上传到 Amazon S3 存储桶。
- 使用 Amazon Rekognition 识别文件中的三个主要属性（标签）。
- 将 EXIF 和标签信息添加到该区域的 Amazon DynamoDB 表中。

适用于 Rust 的 SDK

从 JPG、JPEG 或 PNG 文件中获取 EXIF 信息，将图像文件上传到 Amazon S3 存储桶，使用 Amazon Rekognition 识别文件中的三个主要属性（Amazon Rekognition 中的标签），然后将 EXIF 和标签信息添加到该区域的 Amazon DynamoDB 表中。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Amazon Rekognition
- Amazon S3

使用 SDK for Rust 的 Route 53 示例

以下代码示例向您展示了如何在 Route 53 中使用适用于 Rust 的 AWS SDK 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`ListHostedZones`

以下代码示例演示了如何使用 `ListHostedZones`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListHostedZones于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Amazon S3 示例

以下代码示例向您展示了如何使用适用于 Rust 的软件开发工具包和 Amazon AWS S3 来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作
- 场景
- 无服务器示例

开始使用

开始使用 Amazon S3

以下代码示例显示了如何开始使用 Amazon S3。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/// S3 Hello World Example using the AWS SDK for Rust.
///
/// This example lists the objects in a bucket, uploads an object to that bucket,
/// and then retrieves the object and prints some S3 information about the object.
/// This shows a number of S3 features, including how to use built-in paginators
/// for large data sets.
///
```

```
/// # Arguments
///
/// * `client` - an S3 client configured appropriately for the environment.
/// * `bucket` - the bucket name that the object will be uploaded to. Must be
 present in the region the `client` is configured to use.
/// * `filename` - a reference to a path that will be read and uploaded to S3.
/// * `key` - the string key that the object will be uploaded as inside the bucket.
async fn list_bucket_and_upload_object(
    client: &aws_sdk_s3::Client,
    bucket: &str,
    filepath: &Path,
    key: &str,
) -> Result<(), S3ExampleError> {
    // List the buckets in this account
    let mut objects = client
        .list_objects_v2()
        .bucket(bucket)
        .into_paginator()
        .send();

    println!("key\tetag\tlast_modified\tstorage_class");
    while let Some(Ok(object)) = objects.next().await {
        for item in object.contents() {
            println!(
                "{}\t{}\t{}\t{}",
                item.key().unwrap_or_default(),
                item.e_tag().unwrap_or_default(),
                item.last_modified()
                    .map(|lm| format!("{lm}"))
                    .unwrap_or_default(),
                item.storage_class()
                    .map(|sc| format!("{sc}"))
                    .unwrap_or_default()
            );
        }
    }

    // Prepare a ByteStream around the file, and upload the object using that
 ByteStream.
    let body = aws_sdk_s3::primitives::ByteStream::from_path(filepath)
        .await
        .map_err(|err| {
            S3ExampleError::new(format!(
                "Failed to create bytestream for {filepath:?} ({err:?})"
```

```
            ))
        })?;
    let resp = client
        .put_object()
        .bucket(bucket)
        .key(key)
        .body(body)
        .send()
        .await?;

    println!(
        "Upload success. Version: {:?}",
        resp.version_id()
            .expect("S3 Object upload missing version ID")
    );

    // Retrieve the just-uploaded object.
    let resp = client.get_object().bucket(bucket).key(key).send().await?;
    println!("etag: {}", resp.e_tag().unwrap_or("(missing)"));
    println!("version: {}", resp.version_id().unwrap_or("(missing)"));

    Ok(())
}
```

S3 ExampleError 实用程序。

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

- 有关 API 的详细信息，请参阅适用ListBuckets于 Rust 的AWS SDK API 参考。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建桶并将文件上载到其中。
- 从桶中下载对象。
- 将对象复制到存储桶中的子文件夹。
- 列出存储桶中的对象。
- 删除存储桶及其对象。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

用于运行场景的二进制 crate 的代码。

```
#![allow(clippy::result_large_err)]

//!  Purpose
//!  Shows how to use the AWS SDK for Rust to get started using
//!  Amazon Simple Storage Service (Amazon S3). Create a bucket, move objects into
 and out of it,
//!  and delete all resources at the end of the demo.
//!
//!  This example follows the steps in "Getting started with Amazon S3" in the
 Amazon S3
//!  user guide.
//!  - https://docs.aws.amazon.com/AmazonS3/latest/userguide/GetStartedWithS3.html

use aws_config::meta::region::RegionProviderChain;
use aws_sdk_s3::{config::Region, Client};
use s3_code_examples::error::S3ExampleError;
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), S3ExampleError> {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
    let region = region_provider.region().await.unwrap();
    let shared_config = aws_config::from_env().region(region_provider).load().await;
    let client = Client::new(&shared_config);
    let bucket_name = format!("amzn-s3-demo-bucket-{}", Uuid::new_v4());
    let file_name = "s3/testfile.txt".to_string();
    let key = "test file key name".to_string();
    let target_key = "target_key".to_string();

    if let Err(e) = run_s3_operations(region, client, bucket_name, file_name, key,
 target_key).await
    {
```

```
        eprintln!("{:?}", e);
    };

    Ok(())
}

async fn run_s3_operations(
    region: Region,
    client: Client,
    bucket_name: String,
    file_name: String,
    key: String,
    target_key: String,
) -> Result<(), S3ExampleError> {
    s3_code_examples::create_bucket(&client, &bucket_name, &region).await?;
    let run_example: Result<(), S3ExampleError> = (async {
        s3_code_examples::upload_object(&client, &bucket_name, &file_name,
 &key).await?;
        let _object = s3_code_examples::download_object(&client, &bucket_name,
 &key).await;
        s3_code_examples::copy_object(&client, &bucket_name, &bucket_name, &key,
 &target_key)
            .await?;
        s3_code_examples::list_objects(&client, &bucket_name).await?;
        s3_code_examples::clear_bucket(&client, &bucket_name).await?;
        Ok(())
    })
    .await;
    if let Err(err) = run_example {
        eprintln!("Failed to complete getting-started example: {err:?}");
    }
    s3_code_examples::delete_bucket(&client, &bucket_name).await?;

    Ok(())
}
```

场景使用的常用操作。

```
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))
            }
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的以下主题。

- CopyObject
- CreateBucket
- DeleteBucket
- DeleteObjects
- GetObject
- ListObjectsV2
- PutObject

操作

`CompleteMultipartUpload`

以下代码示例演示了如何使用 `CompleteMultipartUpload`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用CompleteMultipartUpload于 Rust 的AWS SDK API 参考。

`CopyObject`

以下代码示例演示了如何使用 `CopyObject`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用CopyObject于 Rust 的AWS SDK API 参考。

`CreateBucket`

以下代码示例演示了如何使用 `CreateBucket`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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))
        }
    })
}
```

- 有关 API 的详细信息，请参阅适用CreateBucket于 Rust 的AWS SDK API 参考。

`CreateMultipartUpload`

以下代码示例演示了如何使用 `CreateMultipartUpload`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    // 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?;
```

- 有关 API 的详细信息，请参阅适用CreateMultipartUpload于 Rust 的AWS SDK API 参考。

`DeleteBucket`

以下代码示例演示了如何使用 `DeleteBucket`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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))
            }
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteBucket于 Rust 的AWS SDK API 参考。

`DeleteObject`

以下代码示例演示了如何使用 `DeleteObject`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DeleteObject于 Rust 的AWS SDK API 参考。

`DeleteObjects`

以下代码示例演示了如何使用 `DeleteObjects`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DeleteObjects于 Rust 的AWS SDK API 参考。

`GetBucketLocation`

以下代码示例演示了如何使用 `GetBucketLocation`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用GetBucketLocation于 Rust 的AWS SDK API 参考。

`GetObject`

以下代码示例演示了如何使用 `GetObject`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用GetObject于 Rust 的AWS SDK API 参考。

ListBuckets

以下代码示例演示了如何使用 ListBuckets。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListBuckets于 Rust 的AWS SDK API 参考。

`ListObjectVersions`

以下代码示例演示了如何使用 `ListObjectVersions`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListObjectVersions于 Rust 的AWS SDK API 参考。

`ListObjectsV2`

以下代码示例演示了如何使用 `ListObjectsV2`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用于 Rust 的AWS SDK 中的 ListObjectsV2 API 参考。

PutObject

以下代码示例演示了如何使用 PutObject。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
}
```

- 有关 API 的详细信息，请参阅适用PutObject于 Rust 的AWS SDK API 参考。

UploadPart

以下代码示例演示了如何使用 UploadPart。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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?;
```

- 有关 API 的详细信息，请参阅适用UploadPart于 Rust 的AWS SDK API 参考。

场景

将文本转换为语音以及将语音转换回文本

以下代码示例展示了如何：

- 使用 Amazon Polly 将纯文本 (UTF-8) 输入文件合成为音频文件。
- 将音频文件上传到 Amazon S3 存储桶。
- 使用 Amazon Transcribe 将音频文件转换为文本。
- 显示文本。

适用于 Rust 的 SDK

使用 Amazon Polly 将纯文本（UTF-8）输入文件合成为音频文件，将音频文件上传到 Amazon S3 存储桶，使用 Amazon Transcribe 将该音频文件转换为文本，然后显示文本。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Polly
- Amazon S3
- Amazon Transcribe

创建预签名 URL

以下代码示例演示了如何为 Amazon S3 创建预签名 URL 以及如何上传对象。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建预签名请求来 GET S3 对象。

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

创建预签名请求来 PUT S3 对象。

```
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())
}
```

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 Rust 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

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

检测图像中的人脸

以下代码示例展示了如何：

- 将图像保存到 Amazon S3 存储桶中。
- 使用 Amazon Rekognition 检测面部细节，例如年龄范围、性别和情绪（如微笑）。
- 显示这些细节。

适用于 Rust 的 SDK

将图像保存到具有 uploads 前缀的 Amazon S3 存储桶中，使用 Amazon Rekognition 检测面部细节，例如年龄范围、性别和情绪（微笑等），并显示这些细节。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3

如果对象已修改，则从桶中获取该对象

下面的代码示例显示如何从 S3 存储桶中的对象读取数据，但前提是该桶自上次检索以来未被修改。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用GetObject于 Rust 的AWS SDK API 参考。

保存 EXIF 和其他图像信息

以下代码示例展示了如何：

- 从 JPG、JPEG 或 PNG 文件中获取 EXIF 信息。
- 将图像文件上传到 Amazon S3 存储桶。
- 使用 Amazon Rekognition 识别文件中的三个主要属性（标签）。
- 将 EXIF 和标签信息添加到该区域的 Amazon DynamoDB 表中。

适用于 Rust 的 SDK

从 JPG、JPEG 或 PNG 文件中获取 EXIF 信息，将图像文件上传到 Amazon S3 存储桶，使用 Amazon Rekognition 识别文件中的三个主要属性（Amazon Rekognition 中的标签），然后将 EXIF 和标签信息添加到该区域的 Amazon DynamoDB 表中。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Amazon Rekognition
- Amazon S3

使用 SDK 进行单元测试和集成测试

以下代码示例显示了如何举例说明使用 AWS SDK 编写单元测试和集成测试时的最佳实践技术。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Cargo.toml 用于测试示例。

```
[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"
```

使用 automock 和服务包装器的单元测试示例。

```
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);
    }
}
```

使用集成测试示例 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(&[]);
    }
}
```

上传或下载大文件

下面的代码示例展示了如何向 Amazon S3 上传大文件或从 Amazon S3 下载大文件。

有关更多信息，请参阅使用分段上传操作上传对象。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

无服务器示例

通过 Amazon S3 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收通过将对象上传到 S3 桶而触发的事件。该函数从事件参数中检索 S3 存储桶名称和对象密钥，并调用 Amazon S3 API 来检索和记录对象的内容类型。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Rust 将 S3 事件与 Lambda 结合使用。

```
// 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 使用 Rust 版 SDK 的人工智能示例

以下代码示例向您展示了如何使用带有 SageMaker AI 的 Rust AWS SDK 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`ListNotebookInstances`

以下代码示例演示了如何使用 `ListNotebookInstances`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListNotebookInstances于 Rust 的AWS SDK API 参考。

`ListTrainingJobs`

以下代码示例演示了如何使用 `ListTrainingJobs`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListTrainingJobs于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Secrets Manager 示例

以下代码示例向您展示了如何使用带有 Secrets Manager 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`GetSecretValue`

以下代码示例演示了如何使用 `GetSecretValue`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用GetSecretValue于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Amazon SES API v2 示例

以下代码示例向您展示了如何使用适用于 Rust 的软件开发工具包和 Amazon AWS SES API v2 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 场景

操作

`CreateContact`

以下代码示例演示了如何使用 `CreateContact`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CreateContact于 Rust 的AWS SDK API 参考。

`CreateContactList`

以下代码示例演示了如何使用 `CreateContactList`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CreateContactList于 Rust 的AWS SDK API 参考。

`CreateEmailIdentity`

以下代码示例演示了如何使用 `CreateEmailIdentity`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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)),
            },
        }
```

- 有关 API 的详细信息，请参阅适用CreateEmailIdentity于 Rust 的AWS SDK API 参考。

`CreateEmailTemplate`

以下代码示例演示了如何使用 `CreateEmailTemplate`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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)),
            },
        }
```

- 有关 API 的详细信息，请参阅适用CreateEmailTemplate于 Rust 的AWS SDK API 参考。

DeleteContactList

以下代码示例演示了如何使用 DeleteContactList。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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}")),
        }
```

- 有关 API 的详细信息，请参阅适用DeleteContactList于 Rust 的AWS SDK API 参考。

`DeleteEmailIdentity`

以下代码示例演示了如何使用 `DeleteEmailIdentity`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
            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));
                }
            }
```

- 有关 API 的详细信息，请参阅适用DeleteEmailIdentity于 Rust 的AWS SDK API 参考。

`DeleteEmailTemplate`

以下代码示例演示了如何使用 `DeleteEmailTemplate`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 的详细信息，请参阅适用DeleteEmailTemplate于 Rust 的AWS SDK API 参考。

GetEmailIdentity

以下代码示例演示了如何使用 GetEmailIdentity。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

确定电子邮件地址是否已验证。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用GetEmailIdentity于 Rust 的AWS SDK API 参考。

ListContactLists

以下代码示例演示了如何使用 ListContactLists。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListContactLists于 Rust 的AWS SDK API 参考。

`ListContacts`

以下代码示例演示了如何使用 `ListContacts`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListContacts于 Rust 的AWS SDK API 参考。

SendEmail

以下代码示例演示了如何使用 SendEmail。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

向联系人列表中的所有成员发送消息。

```
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(())
}
```

使用模板向联系人列表中的所有成员发送消息。

```
            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)),
            }
```

- 有关 API 的详细信息，请参阅适用SendEmail于 Rust 的AWS SDK API 参考。

场景

时事通讯场景

以下代码示例演示如何运行 Amazon SES API v2 时事通讯场景。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的以下主题。
  - CreateContact
  - CreateContactList
  - CreateEmailIdentity
  - CreateEmailTemplate
  - DeleteContactList

- DeleteEmailIdentity
- DeleteEmailTemplate
- ListContacts
- SendEmail。simple
- SendEmail。模板

使用 SDK for Rust 的 Amazon SNS 示例

以下代码示例向您展示了如何使用适用于 Rust 的软件开发工具包和 Amazon AWS SNS 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 场景
- 无服务器示例

操作

`CreateTopic`

以下代码示例演示了如何使用 `CreateTopic`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用CreateTopic于 Rust 的AWS SDK API 参考。

`ListTopics`

以下代码示例演示了如何使用 `ListTopics`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ListTopics于 Rust 的AWS SDK API 参考。

`Publish`

以下代码示例演示了如何使用 `Publish`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Publish。

`Subscribe`

以下代码示例演示了如何使用 `Subscribe`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

将电子邮件地址订阅到主题。

```
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(())
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API Reference》中的 Subscribe。

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 Rust 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

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

无服务器示例

通过 Amazon SNS 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 主题的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 SNS 事件与 Lambda 结合使用。

```
// 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
}
```

使用 SDK for Rust 的 Amazon SQS 示例

以下代码示例向您展示了如何使用带有 Amazon SQS 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 无服务器示例

操作

`ListQueues`

以下代码示例演示了如何使用 `ListQueues`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

检索该区域中列出的第一个 Amazon SQS 队列。

```
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())
}
```

- 有关 API 的详细信息，请参阅适用ListQueues于 Rust 的AWS SDK API 参考。

`ReceiveMessage`

以下代码示例演示了如何使用 `ReceiveMessage`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用ReceiveMessage于 Rust 的AWS SDK API 参考。

SendMessage

以下代码示例演示了如何使用 SendMessage。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用SendMessage于 Rust 的AWS SDK API 参考。

无服务器示例

通过 Amazon SQS 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 队列的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 SQS 事件与 Lambda 结合使用。

```
// 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
}
```

报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

以下代码示例展示了如何为接收来自 SQS 队列的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Rust 进行 Lambda SQS 批处理项目失败。

```
// 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 使用 Rust 版 SDK 的示例

以下代码示例向您展示了如何使用适用于 Rust 的 AWS SDK 来执行操作和实现常见场景 AWS STS。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`AssumeRole`

以下代码示例演示了如何使用 `AssumeRole`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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),
    }
}
```

- 有关 API 的详细信息，请参阅适用AssumeRole于 Rust 的AWS SDK API 参考。

使用 SDK for Rust 的 Systems Manager 示例

以下代码示例向您展示了如何使用带有 Systems Manager 的 Rust AWS 开发工具包来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`DescribeParameters`

以下代码示例演示了如何使用 `DescribeParameters`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeParameters于 Rust 的AWS SDK API 参考。

`GetParameter`

以下代码示例演示了如何使用 `GetParameter`。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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)
    }
```

- 有关 API 的详细信息，请参阅适用GetParameter于 Rust 的AWS SDK API 参考。

PutParameter

以下代码示例演示了如何使用 PutParameter。

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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(())
}
```

- 有关 API 的详细信息，请参阅适用PutParameter于 Rust 的AWS SDK API 参考。

使用适用于 Rust 的 SDK 的 Amazon Transcribe 示例

以下代码示例向您展示了如何使用带有 Amazon Transcribe 的 Rust AWS 开发工具包来执行操作和实现常见场景。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 场景

场景

将文本转换为语音以及将语音转换回文本

以下代码示例展示了如何：

- 使用 Amazon Polly 将纯文本 (UTF-8) 输入文件合成为音频文件。
- 将音频文件上传到 Amazon S3 存储桶。
- 使用 Amazon Transcribe 将音频文件转换为文本。
- 显示文本。

适用于 Rust 的 SDK

使用 Amazon Polly 将纯文本（UTF-8）输入文件合成为音频文件，将音频文件上传到 Amazon S3 存储桶，使用 Amazon Transcribe 将该音频文件转换为文本，然后显示文本。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Polly
- Amazon S3
- Amazon Transcribe

本 AWS 产品或服务的安全性

云安全性一直是 Amazon Web Services（AWS）的重中之重。作为 AWS 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的安全性和云中的安全性。

云安全 — AWS 负责保护运行 AWS 云中提供的所有服务的基础架构，并为您提供可以安全使用的服务。我们的安全责任是重中之重 AWS，作为AWS 合规计划的一部分，第三方审计师定期测试和验证我们安全的有效性。

云端安全 — 您的责任由您使用的 AWS 服务以及其他因素决定，包括数据的敏感性、组织的要求以及适用的法律和法规。

本 AWS 产品或服务通过其支持的特定 Amazon Web Services (AWS) 服务遵循分担责任模式。有关 AWS 服务安全信息，请参阅AWS 服务安全文档页面和合规计划合 AWS 规工作范围内的AWS 服务。

主题

- 本 AWS 产品或服务中的数据保护
- 此 AWS 产品或服务的合规性验证
- 本 AWS 产品或服务的基础设施安全
- 在中强制使用最低 TLS 版本 适用于 Rust 的 AWS SDK

本 AWS 产品或服务中的数据保护

分 AWS 担责任模型适用于本 AWS 产品或服务中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。

- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》https://aws.amazon.com/compliance/fips/。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API 或 AWS 服务 使用本 AWS 产品或服务或其他产品或服务时 AWS SDKs。 AWS CLI在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

此 AWS 产品或服务的合规性验证

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。有关您在使用时的合规责任的更多信息 AWS 服务，请参阅AWS 安全文档。

本 AWS 产品或服务通过其支持的特定 Amazon Web Services (AWS) 服务遵循分担责任模式。有关 AWS 服务安全信息，请参阅AWS 服务安全文档页面和合规计划合 AWS 规工作范围内的AWS 服务。

本 AWS 产品或服务的基础设施安全

本 AWS 产品或服务使用托管服务，因此受到 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用通过网络访问此 AWS 产品或服务。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。

- 具有完全向前保密（PFS）的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

本 AWS 产品或服务通过其支持的特定 Amazon Web Services (AWS) 服务遵循分担责任模式。有关 AWS 服务安全信息，请参阅AWS 服务安全文档页面和合规计划合 AWS 规工作范围内的AWS 服务。

在中强制使用最低 TLS 版本 适用于 Rust 的 AWS SDK

在与 AWS 服务通信时， 适用于 Rust 的 AWS SDK 使用 TLS 来提高安全性。默认情况下，SDK 强制实施 TLS 最低版本 1.2。默认情况下，SDK 还会协商客户端应用程序和服务可用的最高 TLS 版本。例如，SDK 可能能够协商 TLS 1.3。

通过手动配置 SDK 使用的 TCP 连接器，可以在应用程序中强制实施特定的 TLS 版本。为了说明这一点，以下示例向您展示了如何强制实施 TLS 1.3。

> **Note**
>
> 某些 AWS 服务尚不支持 TLS 1.3，因此强制使用此版本可能会影响 SDK 的互操作性。我们建议在生产部署之前对每项服务测试此配置。

```
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(())
}
```

适用于 Rust 的 AWS SDK 使用的 crate

本主题包含有关 适用于 Rust 的 AWS SDK 使用的 crate 的高级信息。这包括它使用的 Smithy 组件、在某些构建环境下可能需要使用的 crate 以及其他信息。

Smithy crate

和大多数 AWS SDK 一样，适用于 Rust 的 AWS SDK 基于 Smithy。Smithy 是一种用于描述 SDK 提供的数据类型和函数的语言。然后使用这些模型来帮助构建 SDK 本身。

在查看适用于 Rust 的 SDK crate 版本及其 Smithy 依赖项的版本时，知道这些 crate 都使用标准语义版本编号可能会有所帮助。

有关适用于 Rust 的 Smithy crate 的更多详细信息，请参阅 Smithy Rust 设计。

可与适用于 Rust 的 SDK 结合使用的 crate

AWS 发布了很多 Smithy crate。其中一些与适用于 Rust 的 SDK 用户有关，而另一些则是实现细节：

`aws-smithy-async`

如果您不使用 Tokio 来实现异步功能，请包含此 crate。

`aws-smithy-runtime`

包含所有 AWS SDK 所需的构造块。

`aws-smithy-runtime-api`

SDK 使用的底层接口。

`aws-smithy-types`

从其他 AWS SDK 重新导出的类型。如果您使用多个 SDK，请使用此 crate。

`aws-smithy-types-convert`

用于移入和移出 `aws-smithy-types` 的实用程序函数。

其他 crate

存在以下 crate，但您不必了解它们：

适用于 Rust 的 SDK 用户不需要的与服务器相关的 crate：

- aws-smithy-http-server
- aws-smithy-http-server-python

包含 SDK 用户不需要使用的后台代码的 crate：

- aws-smithy-checksum-callbacks
- aws-smithy-eventstream
- aws-smithy-http
- aws-smithy-protocol-test
- aws-smithy-query
- aws-smithy-json
- aws-smithy-xml

不受支持且将来会消失的 crate：

- aws-smithy-client
- aws-smithy-http-auth
- aws-smithy-http-tower

文档历史记录

本主题介绍《适用于 Rust 的 AWS SDK 开发人员指南》在其发展历程中的重要更改。

变更	说明	日期
单元测试	更新了 SDK 中支持的单元测试选项。	2025 年 5 月 2 日
内容重新组织	更新了目录和内容组织方式以便更好地与其他 AWS SDK 保持一致。	2025 年 4 月 7 日
更新了 HTTP	更新了服务客户端的默认 HTTP 功能。	2025 年 3 月 11 日
重新整理了目录	重新整理了目录以更好地区分配置与用法。	2024 年 7 月 1 日
适用于 Rust 的 AWS SDK 公开发行版	更新了指南，增加了新的安全信息、新的和更新的代码示例、有关单元测试的新详细信息及示例，以及新 SDK 公开发行版的其他新内容和更新内容。	2023 年 11 月 27 日
强制实施最低 TLS 版本	添加了有关如何在 SDK 中强制执行 TLS 版本的信息。	2022 年 5 月 4 日
适用于 Rust 的 AWS SDK 开发人员预览版本	开发人员预览版本	2021 年 12 月 2 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。