

開發人員指南

# 適用於 Rust 的 AWS SDK



# 適用於 Rust 的 AWS SDK: 開發人員指南

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

# Table of Contents

什麼是 適用於 Rust 的 AWS SDK ? .....	1
開發套件入門 .....	1
開發套件主要版本的維護與支援 .....	1
其他資源 .....	1
開始使用 .....	3
使用 進行 SDK 身分驗證 AWS .....	3
啟動 AWS 存取入口網站工作階段 .....	4
更多身分驗證資訊 .....	5
Hello 教學課程 .....	5
必要條件 .....	5
建立您的第一個 SDK 應用程式 .....	5
基本概念 .....	7
必要條件 .....	5
Rust 基本概念 .....	8
適用於 Rust 的 AWS SDK 條板箱基本概念 .....	9
使用的專案組態 AWS 服務 .....	9
Tokio 執行時間 .....	10
設定 .....	11
建立服務用戶端 .....	11
從環境設定用戶端 .....	12
針對服務特定的設定使用建置器模式 .....	13
進階明確用戶端組態 .....	13
行為版本 .....	14
在 中設定行為版本 Cargo.toml .....	15
在程式碼中設定行為版本 .....	15
登入資料提供者 .....	15
登入資料提供者鏈 .....	15
明確登入資料提供者 .....	18
身分快取 .....	18
AWS 區域 選擇 .....	18
AWS 區域 提供者鏈結 .....	18
組態選項 .....	19
用戶端端點 .....	19
自訂組態 .....	20

範例 .....	23
HTTP .....	24
Hyper 1.x .....	24
攔截器 .....	26
攔截器註冊 .....	28
可觀測性 .....	28
日誌 .....	29
覆寫操作組態 .....	32
重試 .....	33
預設重試組態 .....	33
最大嘗試次數 .....	34
延遲和退避 .....	34
自適應重試模式 .....	35
逾時 .....	35
API 逾時 .....	36
停滯的串流保護 .....	37
使用 SDK .....	39
提出請求 .....	39
最佳實務 .....	40
盡可能重複使用 SDK 用戶端 .....	40
設定 API 逾時 .....	40
並行數量 .....	41
條款 .....	41
簡單範例 .....	41
擁有權和可變性 .....	43
更多詞彙！ .....	43
重寫我們的範例，以提升效率（單執行緒並行） .....	44
重寫我們的範例以提高效率（多執行緒並行） .....	46
偵錯多執行緒應用程式 .....	47
錯誤處理 .....	47
服務錯誤 .....	48
錯誤中繼資料 .....	48
使用 列印的詳細錯誤 DisplayErrorContext .....	49
建立 Lambda 函數 .....	51
分頁 .....	51
建立預先簽章URLs .....	53

預先簽章基本概念 .....	53
預先簽章POST和PUT請求 .....	54
獨立簽署者 .....	55
單元測試 .....	56
使用模擬模型自動產生模擬 .....	56
靜態重播 .....	61
等待程式 .....	64
程式碼範例 .....	66
API Gateway .....	67
動作 .....	68
案例 .....	69
AWS 社群貢獻 .....	69
API Gateway Management API .....	70
動作 .....	68
Application Auto Scaling .....	71
動作 .....	68
Aurora .....	72
基本概念 .....	74
動作 .....	68
Auto Scaling .....	220
基本概念 .....	74
動作 .....	68
Amazon Bedrock 執行期 .....	254
Anthropic Claude .....	255
Amazon Cognito 身分提供者 .....	270
動作 .....	68
Amazon Cognito Sync .....	271
動作 .....	68
Firehose .....	273
動作 .....	68
Amazon DocumentDB .....	274
無伺服器範例 .....	274
DynamoDB .....	276
動作 .....	68
案例 .....	69
無伺服器範例 .....	274

AWS 社群貢獻 .....	69
Amazon EBS .....	294
動作 .....	68
Amazon EC2 .....	296
基本概念 .....	74
動作 .....	68
Amazon ECR .....	357
動作 .....	68
Amazon ECS .....	359
動作 .....	68
Amazon EKS .....	362
動作 .....	68
AWS Glue .....	364
基本概念 .....	74
動作 .....	68
IAM .....	379
基本概念 .....	74
動作 .....	68
AWS IoT .....	407
動作 .....	68
Kinesis .....	409
動作 .....	68
無伺服器範例 .....	274
AWS KMS .....	416
動作 .....	68
Lambda .....	425
基本概念 .....	74
動作 .....	68
案例 .....	69
無伺服器範例 .....	274
AWS 社群貢獻 .....	69
MediaLive .....	475
動作 .....	68
MediaPackage .....	476
動作 .....	68
Amazon MSK .....	478

無伺服器範例 .....	274
Amazon Polly .....	480
動作 .....	68
案例 .....	69
QLDB .....	485
動作 .....	68
Amazon RDS .....	486
無伺服器範例 .....	274
Amazon RDS 資料服務 .....	490
動作 .....	68
Amazon Rekognition .....	491
案例 .....	69
Route 53 .....	493
動作 .....	68
Amazon S3 .....	494
基本概念 .....	74
動作 .....	68
案例 .....	69
無伺服器範例 .....	274
SageMaker AI .....	544
動作 .....	68
Secrets Manager .....	547
動作 .....	68
Amazon SES API v2 .....	548
動作 .....	68
案例 .....	69
Amazon SNS .....	564
動作 .....	68
案例 .....	69
無伺服器範例 .....	274
Amazon SQS .....	570
動作 .....	68
無伺服器範例 .....	274
AWS STS .....	575
動作 .....	68
Systems Manager .....	576

---

動作 .....	68
Amazon Transcribe .....	579
案例 .....	69
安全 .....	581
資料保護 .....	581
合規驗證 .....	582
基礎設施安全性 .....	583
強制執行最低 TLS 版本 .....	583
軟體開發套件使用的板條箱 .....	586
Smithy 條板箱 .....	586
與 SDK 搭配使用的條板箱 .....	586
其他條板箱 .....	586
文件歷史紀錄 .....	588
.....	dlxxxix



# 什麼是適用於 Rust 的 AWS SDK？

Rust 是一種系統程式設計語言，沒有專注於三個目標的垃圾收集器：安全、速度和並行。

適用於 Rust 的 AWS SDK（開發套件）提供 Rust APIs 來與 Amazon Web Services 基礎設施服務互動。使用 SDK，您可以在 Amazon S3、Amazon EC2、DynamoDB 等基礎上建置應用程式。

主題

- [開發套件入門](#)
- [開發套件主要版本的維護與支援](#)
- [其他資源](#)

## 開發套件入門

如果您是第一次使用 SDK，建議您先閱讀 [開始使用適用於 Rust 的 AWS SDK](#)。

如需組態和設定，包括如何建立和設定服務用戶端以向發出請求 AWS 服務，請參閱 [設定適用於 Rust 的 AWS SDK](#)。

如需有關使用 SDK 的資訊，請參閱 [使用適用於 Rust 的 AWS SDK](#)。

如需 Rust 程式碼範例的完整清單，請參閱 [程式碼範例](#)。

## 開發套件主要版本的維護與支援

如需開發套件主要版本及其基礎相依性之維護與支援的相關資訊，請參閱 [《AWS 開發套件及工具參考指南》](#) 中的以下內容：

- [AWS SDKs和工具維護政策](#)
- [AWS SDKs和工具版本支援矩陣](#)

## 其他資源

除了本指南之外，以下是 SDK 開發人員的寶貴線上資源：

- [AWS SDKs和工具參考指南](#)：包含設定、功能和其他在 AWS SDKs之間常見的基本概念。

- [Rust 程式設計語言網站](#)
- [適用於 Rust 的 AWS SDK API 參考](#)
- [AWS 開發人員工具部落格](#)
- GitHub 上的[適用於 Rust 的 AWS SDK 原始程式碼](#)
- [AWS 程式碼範例目錄](#)

# 開始使用 適用於 Rust 的 AWS SDK

了解如何安裝、設定和使用 SDK 來建立 Rust 應用程式，以程式設計方式存取 AWS 資源。

## 主題

- [使用 進行 SDK 身分驗證 AWS](#)
- [的 Hello 教學課程 適用於 Rust 的 AWS SDK](#)
- [基本概念](#)

## 使用 進行 SDK 身分驗證 AWS

使用 開發 AWS 時，您必須建立程式碼如何向 進行身分驗證 AWS 服務。您可以根據環境和您可用的存取權，以不同的方式設定 AWS 資源的程式設計 AWS 存取。

若要選擇您的身分驗證方法並針對 SDK 設定，請參閱 AWS SDKs和工具參考指南中的[身分驗證和存取](#)。

我們建議在本機開發且未獲得其雇主身分驗證方法的新使用者應進行設定 AWS IAM Identity Center。此方法包括安裝 AWS CLI 以方便設定，以及定期登入 AWS 存取入口網站。如果您選擇此方法，您的環境應該在您完成 AWS SDKs和工具參考指南中的[IAM Identity Center 身分驗證](#)程序之後包含下列元素：

- 在執行應用程式之前 AWS CLI，您用來啟動 AWS 存取入口網站工作階段的。
- 共用[AWSconfig](#)檔案，其[default]設定檔具有一組可從 SDK 參考的組態值。若要尋找此檔案的位置，請參閱 AWS SDK 和工具參考指南中的[共用檔案位置](#)。
- 共用config檔案會設定 [region](#)設定。這會設定 SDK 用於 AWS 請求 AWS 區域 的預設值。此區域用於未指定使用 區域的 SDK 服務請求。
- 軟體開發套件使用描述檔的 [SSO 字符提供者組態](#)，在傳送請求至 之前取得憑證 AWS。此sso\_role\_name值是連接至 IAM Identity Center 許可集的 IAM 角色，允許存取應用程式中 AWS 服務 使用的。

下列範例config檔案顯示使用 SSO 字符提供者組態設定的預設設定檔。設定檔sso\_session的設定是指具名[sso-session](#)區段。sso-session 本節包含啟動 AWS 存取入口網站工作階段的設定。

```
[default]
```

```
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

適用於 Rust 的 SDK 不需要將其他套件（例如 SSO 和 SSO0IDC）新增至您的應用程式，即可使用 IAM Identity Center 身分驗證。

## 啟動 AWS 存取入口網站工作階段

在執行存取的應用程式之前 AWS 服務，您需要 SDK 的作用中 AWS 存取入口網站工作階段，才能使用 IAM Identity Center 身分驗證來解析登入資料。視您設定的工作階段長度而定，您的存取最終會過期，且 SDK 將遇到身分驗證錯誤。若要登入 AWS 存取入口網站，請在 中執行下列命令 AWS CLI。

```
$ aws sso login
```

如果您遵循指引並設定預設設定檔，則不需要使用 `--profile` 選項呼叫 命令。如果您的 SSO 權杖提供者組態使用已命名的設定檔，則命令為 `aws sso login --profile named-profile`。

若要選擇性地測試您是否已經有作用中的工作階段，請執行下列 AWS CLI 命令。

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

如果您的工作階段處於作用中狀態，對此命令的回應會報告共用 config 檔案中設定的 IAM Identity Center 帳戶和許可集。

### Note

如果您已有作用中的 AWS 存取入口網站工作階段並執行 `aws sso login`，則不需要提供登入資料。

登入程序可能會提示您允許 AWS CLI 存取您的資料。由於 AWS CLI 建置在適用於 Python 的 SDK 之上，因此許可訊息可能包含 `botocore` 名稱的變化。

## 更多身分驗證資訊

人類使用者具有人類身分，是應用程式的相關人員、管理員、開發人員、操作員和消費者。它們必須具有身分才能存取您的 AWS 環境和應用程式。屬於您組織成員的人類使用者 - 這表示您是開發人員 - 稱為人力身分。

存取時使用臨時憑證 AWS。您可以使用身分提供者，讓您的人類使用者透過擔任角色來提供 AWS 帳戶的聯合存取，而該角色提供臨時登入資料。對於集中式存取管理，我們建議您使用 AWS IAM Identity Center (IAM Identity Center) 來管理對這些帳戶中的帳戶和許可的存取。如需更多替代方案，請參閱以下內容：

- 如需了解有關最佳實務的資訊，請參閱 IAM 使用者指南中的 [IAM 安全最佳實務](#)。
- 若要建立短期 AWS 登入資料，請參閱《IAM 使用者指南》中的 [臨時安全登入](#) 資料。
- 若要了解 SDK for Rust 支援的其他憑證提供者，請參閱 AWS SDKs 和工具參考指南中的 [標準化憑證提供者](#)。

## 的 Hello 教學課程 適用於 Rust 的 AWS SDK

### 必要條件

若要使用 適用於 Rust 的 AWS SDK，您必須安裝 Rust 和 Cargo。

- 安裝 Rust 工具鏈：<https://www.rust-lang.org/tools/install>
- 透過執行命令來安裝 cargo-component [工具](#)：`cargo install cargo-component`

### 建議的工具：

下列選用工具可以安裝在 IDE 中，以協助程式碼完成和故障診斷。

- rust-analyzer 延伸模組，請參閱 [Visual Studio Code 中的 Rust](#)。
- Amazon Q Developer，請參閱 [在您的 IDE 中安裝 Amazon Q Developer 延伸模組或外掛程式](#)。

### 建立您的第一個 SDK 應用程式

此程序會建立第一個列出 DynamoDB 資料表的 Rust 專用開發套件應用程式。

1. 在終端機或主控台視窗中，導覽至電腦上您要建立應用程式的位置。

- 執行下列命令來建立hello\_world目錄，並將其填入骨架 Rust 專案：

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

- 導覽至 hello\_world目錄，並使用下列命令將所需的相依性新增至應用程式：

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

這些相依性包括提供 DynamoDB 組態功能和支援的 SDK 板條箱，包括用於實作非同步 I/O 操作的[tokio板條箱](#)。

#### Note

除非您使用像 tokio/full Tokio 這樣的功能，否則不會提供非同步執行時間。適用於 Rust 的 SDK 需要非同步執行時間。

- 在 main.rs src目錄中更新 以包含下列程式碼。

```
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

此範例只會顯示結果的第一頁。請參閱 [以 the section called “分頁”](#) 了解如何處理多頁的結果。

## 5. 執行程式：

```
$ cargo run
```

您應該會看到資料表名稱的清單。

## 基本概念

### 必要條件

若要使用 適用於 Rust 的 AWS SDK，您必須安裝 Rust 和 Cargo。

- 安裝 Rust 工具鏈：<https://www.rust-lang.org/tools/install>
- 透過執行命令來安裝 cargo-component [工具](#)：`cargo install cargo-component`

### 建議的工具：

下列選用工具可以安裝在您的 IDE 中，以協助程式碼完成和故障診斷。

- rust-analyzer 延伸模組，請參閱 [Visual Studio Code 中的 Rust](#)。
- Amazon Q Developer，請參閱 [在您的 IDE 中安裝 Amazon Q Developer 延伸模組或外掛程式](#)。

## Rust 基本概念

以下是 Rust 程式設計語言的一些基本概念，有助於了解。所有詳細資訊的參考皆來自 [Rust 程式設計語言](#)。

- Cargo.toml 是標準 Rust 專案組態檔案，其中包含相依性和專案的一些中繼資料。Rust 來源檔案具有 .rs 副檔名。請參閱 [Hello, Cargo!](#)。
- Cargo.toml 您可以使用設定檔自訂，請參閱[使用發行設定檔自訂建置](#)。這些設定檔與共用 AWS config 檔案中的設定檔 AWS 使用完全無關且獨立。
- 將程式庫相依性新增至專案和此檔案的常見方法是使用 cargo add。請參閱 [cargo-add](#)。
- Rust 的基本函數結構如下。let 關鍵字會宣告變數，並可能與指派 (=) 配對。如果您未在之後指定類型 let，則編譯器會推斷類型。請參閱[變數和可變性](#)。

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

- 若要宣告 x 具有明確類型的變數 T，Rust 會使用語法 x: T。請參閱[資料類型](#)。
- struct X {} 會定義新的類型 X。方法實作在自訂結構類型上 X。類型的方法 X 會宣告為實作區塊字首為關鍵字 impl。在實作區塊中，self 是指呼叫方法的结构執行個體。請參閱[關鍵字 impl 和方法語法](#)。
- 如果驚嘆號 (!) 遵循看起來是函數定義或函數呼叫的內容，則程式碼正在定義或呼叫巨集。請參閱[巨集](#)。
- 在 Rust 中，無法復原的錯誤由 panic! 巨集表示。當程式遇到 panic! 會停止執行的時，請列印失敗訊息、展開、清除堆疊，然後結束。請參閱[使用無法復原的錯誤 panic!](#)。
- Rust 不支援像其他程式設計語言一樣從基礎類別繼承功能；Rust 提供方法超載 traits 的方式。特徵可能會被視為概念上類似於 界面。不過，特徵和真實界面有差異，且通常在設計過程中使用不同。請參閱[特徵：定義共用行為](#)。
  - 多態性是指支援多種資料類型功能的程式碼，而不必個別寫入每個資料類型。Rust 透過列舉、特徵和學名藥支援多態性。請參閱[以類型系統和程式碼共用的形式繼承](#)。
- Rust 對記憶體非常明確。智慧型指標「是像指標一樣但也具有其他中繼資料和功能的資料結構」。請參閱[智慧指標](#)。
  - 類型 Cow 是 clone-on-write 的智慧型指標，有助於在必要時將記憶體擁有權轉移給發起人。請參閱 [Enum std::borrow::Cow](#)。



- 類型 `Arc` 是原子參考計數智慧指標，可計算已配置的執行個體。請參閱 [Struct `std::sync::Arc`](#)。
- 適用於 Rust 的 SDK 經常使用建置器模式來建構複雜類型。

## 適用於 Rust 的 AWS SDK 條板箱基本概念

- 適用於 Rust 的 SDK 程式庫會依每個程式庫組分成不同的程式庫組。AWS 服務這些板條箱可在 <https://docs.rs/> 取得。
- AWS 服務 條板箱遵循 的命名慣例 `aws-sdk-[servicename]`，例如 `aws-sdk-s3` 和 `aws-sdk-dynamodb`。
- SDK for Rust 功能的主要核心架構為 `aws-config`。這包含在大多數專案中，因為它提供從環境讀取組態的功能。
  - 請勿將此與名為 AWS 服務的 混淆 AWS Config。由於這是一項 服務，因此會遵循標準慣例並稱為 `aws-sdk-config`。

## 使用的專案組態 AWS 服務

- 您需要為應用程式要使用的每個項目 AWS 服務，將條板箱新增至專案。
- 新增條板箱的建議方法是透過執行 `cargo add [crateName]`，例如 `cargo add aws-sdk-s3`。
  - 這會在 `Cargo.toml` 下將一行新增至專案的 `[dependencies]`。
  - 根據預設，這會將最新版的 條板箱新增至您的專案。
- 在您的來源檔案中，使用 `use` 陳述式將來自其箱子的項目帶入範圍。請參閱 Rust 程式設計語言網站上的 [使用外部套件](#)。
  - 條板箱名稱通常為連字號，但連字號會在實際使用條板箱時轉換為底線。例如，在程式碼 `use` 陳述式中會使用 `aws-config` 條板箱做為 `: use aws_config`。
- 組態是一個複雜的主題。組態可以直接在程式碼中發生，也可以在環境變數或組態檔案中外部指定。如需詳細資訊，請參閱 [組態選項](#)。
  - 當 SDK 載入您的組態時，會記錄無效的值，而不是停止執行，因為大多數設定都有合理的預設值。若要了解如何開啟記錄，請參閱 [啟用 適用於 Rust 的 AWS SDK 程式碼記錄](#)。
  - 大多數環境變數和組態檔案設定會在程式啟動時載入一次。在您重新啟動程式之前，將不會看到任何值的更新。

## Tokio 執行時間

- Tokio 是適用於 Rust 的 SDK 程式設計語言的非同步執行時間，它會執行 `async` 任務。請參閱 [tokio.rs](https://tokio.rs) 和 [docs.rs/tokio](https://docs.rs/tokio)。
- 適用於 Rust 的 SDK 需要非同步執行時間。我們建議您將下列條板箱新增至您的專案：

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

- `tokio::main` 屬性巨集會為您的程式建立非同步主要進入點。若要使用此巨集，請將其新增至 `main` 方法之前的行，如下所示：

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

# 設定 適用於 Rust 的 AWS SDK

了解如何設定 適用於 Rust 的 AWS SDK。當您使用 開發 AWS 時，您必須建立程式碼如何向 進行身分驗證 AWS 服務。您也必須設定 AWS 區域 要使用的。

[AWS SDKs和工具參考指南](#) 也包含許多 AWS SDKs 中常見的設定、功能和其他基本概念。

## 主題

- [建立服務用戶端](#)
- [行為版本](#)
- [登入資料提供者](#)
- [AWS 區域 選擇](#)
- [組態選項](#)
- [用戶端端點](#)
- [HTTP](#)
- [攔截器](#)
- [使用可觀測性功能](#)
- [覆寫用戶端的單一操作組態](#)
- [重試](#)
- [逾時](#)

## 建立服務用戶端

若要向 提出請求 AWS 服務，您首先會執行個體化該服務的用戶端。您可以設定服務用戶端的常見設定，例如逾時、HTTP 用戶端和重試組態。

每個服務用戶端都需要 AWS 區域 和 登入資料提供者。SDK 使用這些值將請求傳送到 資源的正確區域，並使用正確的登入資料簽署請求。您可以以程式設計方式在程式碼中指定這些值，或從環境中自動載入這些值。

### Note

服務用戶端建構成本可能很昂貴，通常需要共用。為了促進這一點，所有Client結構都會實作 Clone。

軟體開發套件有一系列位置（或來源），其會進行檢查，以尋找組態設定的值。

1. 程式碼中或服務用戶端本身上設定的任何明確設定，都優先於任何其他設定。
2. 環境變數
  - 如需設定環境變數的詳細資訊，請參閱 AWS SDKs 和工具參考指南中的[環境變數](#)。
3. 共用 config 和 credentials 檔案
  - 如需設定這些檔案的詳細資訊，請參閱 SDK [config](#) 和 [工具參考指南中的共用和 credentials 檔案](#)。AWS SDKs
4. 任何 SDK 原始碼本身提供的預設值都會最後使用。
  - 有些屬性，例如區域，沒有預設值。您必須在程式碼、環境設定或共用 config 檔案中明確指定它們。如果 SDK 無法解析必要的組態，API 請求可能會在執行時間失敗。

大多數環境變數設定和 config 和 credentials 檔案設定是由 AWS SDKs 和工具共用，以實現一致行為。若要查看開發套件可從環境變數或組態檔案解析的所有設定，請參閱開發套件和工具參考指南中的[設定](#)參考。AWS SDKs

## 從環境設定用戶端

若要建立具有環境來源組態的用戶端，請使用 aws-config 條板箱中的靜態方法：

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

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

以這種方式建立用戶端在 Amazon Elastic Compute Cloud 上執行時很有用 AWS Lambda，或可直接從環境中使用服務用戶端組態的任何其他內容。這會將您的程式碼與執行環境分離，讓您更輕鬆地將應用程式部署到多個 AWS 區域而無需變更程式碼。

您可以明確覆寫特定屬性。明確組態優先於從執行環境解析的組態。下列範例會從環境載入組態，但明確覆寫 AWS 區域：

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

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

### Note

並非所有組態值都是由用戶端在建立時取得。當用戶端用於提出請求時，登入資料提供者層會存取登入資料相關設定，例如臨時存取金鑰和 IAM Identity Center 組態。

先前範例中 `BehaviorVersion::latest()` 顯示的程式碼指出要用於預設值的 SDK 版本。`BehaviorVersion::latest()` 適用於大多數情況。如需詳細資訊，請參閱 [行為版本](#)。

## 針對服務特定的設定使用建置器模式

有些選項只能在特定服務用戶端類型上設定。不過，通常您仍然想要從環境中載入大部分的組態，然後特別新增其他選項。建置器模式是適用於 Rust 的 AWS SDK 條板箱內的常見模式。您首先使用載入一般組態 `aws_config::defaults`，然後使用 `from` 方法將該組態載入您正在使用的服務的建置器。然後，您可以為該服務設定任何唯一的組態值，並呼叫 `build`。最後，會從此修改後的組態建立用戶端。

```
// 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);
```

探索特定服務用戶端類型可用的其他方法的一種方法是使用 API 文件，例如的 [aws\\_sdk\\_s3::config::Builder](#)。

## 進階明確用戶端組態

若要設定具有特定值的服務用戶端，而不是從環境載入組態，您可以在用戶端 `Config` 建置器上指定它們，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);
```

當您使用 建立服務組態時`aws_sdk_s3::Config::builder()`，不會載入任何預設組態。預設值只會在根據 建立組態時載入`aws_config::defaults`。

有些選項只能在特定服務用戶端類型上設定。上一個範例顯示使用 Amazon S3 用戶端上的 `endpoint_resolver` 函數，藉此範例。

## 行為版本

適用於 Rust 的 AWS SDK 開發人員預期並依賴語言及其主要程式庫提供的強大且可預測的行為。為了協助開發人員使用適用於 Rust 的 SDK 取得預期的行為，用戶端組態需要包含 `BehaviorVersion`。`BehaviorVersion` 指定預設的 SDK 版本。這可讓 SDK 隨著時間演進，變更最佳實務以符合新標準，並支援新功能，而不會對您應用程式的行為造成非預期的負面影響。

### Warning

如果您嘗試在未明確指定的情況下設定 SDK 或建立用戶端 `BehaviorVersion`，建構器將會 panic。

例如，假設新版本的 SDK 已發行，並具有新的預設重試政策。如果您的應用程式使用 `BehaviorVersion` 符合舊版 SDK 的，則會使用先前的組態，而非新的預設組態。

每次發行適用於 Rust 的新行為版本時，前一個 `BehaviorVersion` 版本都會以適用於 Rust 的 SDK `deprecated` 屬性標記，並新增新版本。這會導致在編譯時間發生警告，但否則可讓組建繼續照常進行。`BehaviorVersion::latest()` 也會更新，以指出新版本的預設行為。

### Note

在大多數情況下，您應該 `BehaviorVersion::latest()` 在程式碼中使用，或在 `Cargo.toml` `behavior-version-latest` 檔案中使用 功能旗標。建議僅在需要時才將 釘選到特定版本。

## 在中設定行為版本 Cargo.toml

您可以在 Cargo.toml 檔案中包含適當的功能旗標，以指定 SDK 和個別模組的行為版本aws-sdk-iam，例如 aws-sdk-s3或。目前，僅支援 SDK 的latest版本Cargo.toml：

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

## 在程式碼中設定行為版本

您的程式碼可以視需要變更行為版本，方法是在設定 SDK 或用戶端時加以指定：

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

此範例會建立使用環境來設定 SDK 的組態，但會將 BehaviorVersion設定為 v2023\_11\_09()。

## 登入資料提供者

若要 AWS 使用 向 發出請求 適用於 Rust 的 AWS SDK，開發套件會使用 發行的密碼編譯簽署憑證 AWS。在執行時間，軟體開發套件會檢查多個位置，以擷取登入資料的組態值。

如果擷取的組態包含[AWS IAM Identity Center 單一登入存取設定](#)，則 SDK 會與 IAM Identity Center 搭配使用，以擷取用於提出請求的臨時憑證 AWS 服務。

如果擷取的組態包含[臨時登入資料](#)，則 SDK 會使用它們來 AWS 服務 呼叫。臨時登入資料包含存取金鑰和工作階段權杖。

使用 進行身分驗證 AWS 可以在您的程式碼庫之外處理。開發套件可以使用登入資料提供者鏈自動偵測、使用和重新整理許多身分驗證方法。

如需專案身分 AWS 驗證入門的引導選項，請參閱 AWS SDKs和工具參考指南中的[身分驗證和存取](#)。

## 登入資料提供者鏈

如果您在建構用戶端時未明確指定登入資料提供者，適用於 Rust 的 SDK 會使用登入資料提供者鏈來檢查一系列可以提供登入資料的位置。一旦 SDK 在其中一個位置找到登入資料，搜尋就會停止。如需建構用戶端的詳細資訊，請參閱[建立服務用戶端](#)。

下列範例不會在程式碼中指定登入資料提供者。SDK 使用登入資料提供者鏈來偵測已在託管環境中設定的身分驗證，並使用該身分驗證來進行呼叫 AWS 服務。

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

## 登入資料擷取順序

登入資料提供者鏈結會使用下列預先定義的順序搜尋登入資料：

### 1. 存取金鑰環境變數

SDK 會嘗試從 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 以及 `AWS_SESSION_TOKEN` 環境變數載入登入資料。

### 2. 共用 AWS `config` 和 `credentials` 檔案

SDK 會嘗試從共用 AWS `config` 和 `credentials` 檔案的 `[default]` 設定檔載入登入資料。您可以使用 `AWS_PROFILE` 環境變數，選擇您希望 SDK 載入的具名設定檔，而不是使用 `[default]`。`config` 和 `credentials` 檔案由各種 AWS SDKs 和工具共用。如需這些檔案的詳細資訊，請參閱 SDK [config](#) 和 [工具參考指南中的共用和 credentials 檔案](#)。AWS SDKs

如果您使用 IAM Identity Center 進行身分驗證，這是指 SDK for Rust 使用執行 CLI AWS 命令所設定的單一登入字符 `aws sso login`。SDK 使用 IAM Identity Center 交換為有效字符的臨時憑證。然後，軟體開發套件會在呼叫時使用臨時登入資料 AWS 服務。如需此程序的詳細資訊，請參閱 [開發套件和工具參考指南中的了解的 SDK 憑證解析 AWS 服務](#)。AWS SDKs

- 如需設定此提供者的指引，請參閱 AWS SDK 和工具參考指南中的 SDKs [IAM Identity Center 身分驗證](#)。
- 如需此提供者 SDK 組態屬性的詳細資訊，請參閱 SDK AWS 和工具參考指南中的 SDKs [IAM Identity Center 登入資料提供者](#)。

### 3. AWS STS Web 身分

建立需要存取的行動應用程式或用戶端型 Web 應用程式時 AWS，AWS Security Token Service (AWS STS) 會傳回一組臨時安全登入資料，供透過公有身分提供者 (IdP) 進行身分驗證的聯合身分使用者使用。

- 當您在設定檔中指定此項目時，軟體開發套件或工具會嘗試使用 AWS STS `AssumeRoleWithWebIdentity` API 方法擷取臨時登入資料。如需此方法的詳細資訊，請參閱 AWS Security Token Service API 參考中的 [AssumeRoleWithWebIdentity](#)。



- 如需設定此提供者的指引，請參閱 AWS SDKs 和工具參考指南中的 [使用 Web 身分或 OpenID Connect 聯合](#)。
- 如需此供應商的 SDK 組態屬性詳細資訊，請參閱 SDK AWS SDKs 和工具參考指南中的 [擔任角色登入資料供應商](#)。

#### 4. Amazon ECS 和 Amazon EKS 容器憑證

您的 Amazon Elastic Container Service 任務和 Kubernetes 服務帳戶可以具有與其相關聯的 IAM 角色。在 IAM 角色中授予的許可，是由在 Pod 的任務或容器中執行的容器所擔任。此角色可讓您的 SDK for Rust 應用程式碼（在容器上）使用其他 AWS 服務。

軟體開發套件會嘗試從 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` 或 `AWS_CONTAINER_CREDENTIALS_FULL_URI` 環境變數擷取登入資料，這些變數可由 Amazon ECS 和 Amazon EKS 自動設定。

- 如需為 Amazon ECS 設定此角色的詳細資訊，請參閱《Amazon Elastic Container Service 開發人員指南》中的 Amazon [ECS 任務 IAM 角色](#)。
- 如需 Amazon EKS 設定資訊，請參閱《[Amazon EKS 使用者指南](#)》中的設定 [Amazon EKS Pod Identity Agent](#)。
- 如需此提供者 SDK 組態屬性的詳細資訊，請參閱 SDK AWS SDKs 和工具參考指南中的 [容器憑證提供者](#)。

#### 5. Amazon EC2 執行個體中繼資料服務

建立 IAM 角色並將其連接至您的執行個體。執行個體上的適用於 Rust 的 SDK 應用程式會嘗試從執行個體中繼資料擷取角色提供的登入資料。

- SDK for Rust 僅支援 [IMDSv2](#)。
- 如需設定此角色和使用中繼資料的詳細資訊，請參閱《[Amazon EC2 使用者指南](#)》中的適用於 [Amazon EC2](#) 和 [使用執行個體中繼資料](#) 的 IAM 角色。Amazon EC2
- 如需此提供者 SDK 組態屬性的詳細資訊，請參閱 SDK AWS SDKs 和工具參考指南中的 [IMDS 登入資料提供者](#)。

#### 6. 如果目前仍無法解析登入資料，操作會 panics 發生錯誤。

如需 AWS 登入資料提供者組態設定的詳細資訊，請參閱 AWS SDKs 和工具參考指南的設定參考中的 [標準化登入資料提供者](#)。

## 明確登入資料提供者

您可以指定 SDK 應該使用的特定登入資料提供者，而不是依賴登入資料提供者鏈來偵測您的身分驗證方法。當您使用 載入一般組態時 `aws_config::defaults`，您可以指定自訂登入資料提供者，如下所示：

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

您可以透過實作 [ProvideCredentials](#) 特性來實作自己的登入資料提供者。

## 身分快取

軟體開發套件會快取登入資料和其他身分類型，例如 SSO 字符。根據預設，軟體開發套件會使用延遲快取實作，在第一次請求時載入登入資料、快取登入資料，然後在另一個請求接近過期時嘗試重新整理登入資料。從相同 建立的用戶端 `SdkConfig` 將共用 [IdentityCache](#)。

## AWS 區域 選擇

您可以使用 存取在特定地理區域中操作 AWS 服務的 AWS 區域。這對於備援和讓資料和應用程式在接近您和使用者存取它們的位置時運作都很有用。如需如何使用區域的詳細資訊，請參閱 AWS SDKs 和工具參考指南 [AWS 區域](#) 中的。

### Important

大多數資源都位於特定 中，AWS 區域 而且在使用 SDK 時，您必須提供資源的正確區域。

## AWS 區域 提供者鏈結

從執行環境載入服務用戶端的組態時，會使用下列查詢程序。軟體開發套件找到的第一個值用於用戶端的組態。如需建立服務用戶端的詳細資訊，請參閱 [從環境設定用戶端](#)。

1. 任何以程式設計方式設定的明確區域。
2. 檢查 `AWS_REGION` 環境變數。
  - 如果您使用 AWS Lambda 服務，容器會自動設定此環境變數 `AWS Lambda`。
3. 會檢查共用 AWS config 檔案中的 `region` 屬性。

- `AWS_CONFIG_FILE` 環境變數可用來變更共用config檔案的位置。若要進一步了解保留此檔案的位置，請參閱 SDK [config和工具參考指南中的共用和credentials檔案的位置](#)。AWS SDKs
  - `AWS_PROFILE` 環境變數可用來選取具名設定檔，而非預設值。若要進一步了解如何設定不同的設定檔，請參閱 SDK [config和工具參考指南中的共用和credentials檔案](#)。AWS SDKs
4. 軟體開發套件會嘗試使用 Amazon EC2 執行個體中繼資料服務來判斷目前執行中 Amazon EC2 執行個體的區域。
- 適用於 Rust 的 AWS SDK 僅支援 IMDSv2。

## 組態選項

許多組態設定可以在程式碼之外處理。大多數組態設定可以設定為環境變數或單獨的共用 AWS config檔案中。共用config檔案可以維護個別的設定集，稱為設定檔，為不同的環境或測試提供不同的組態。

環境變數和共用config檔案設定是標準化的，並跨 AWS SDKs和工具共用，以支援不同語言的一致功能。

請參閱 AWS SDKs和工具參考指南，了解如何透過這些方法設定您的應用程式，以及每個跨 sdk 設定的詳細資訊。有些感興趣的頁面可能是：

- [共用config和credentials檔案](#) - 說明設定檔和這些組態檔案的格式。
- [共用config和credentials檔案的位置](#) - 說明這些檔案的預設位置，以及如何變更檔案。
- [環境變數支援](#) - 說明設定環境變數。
- [設定參考](#) - 所有跨 SDK 設定的參考資訊。

## 用戶端端點

當適用於 Rust 的 AWS SDK 呼叫時 AWS 服務，其第一個步驟是判斷路由請求的位置。此程序稱為端點解析。

您可以在建立服務用戶端時設定 SDK 的端點解析。端點解析的預設組態通常沒問題，但您可能想要修改預設組態有幾個原因。兩個範例原因如下：

- 向服務的預先發行版本或服務的本機部署提出請求。
- 存取 SDK 中尚未建模的特定服務功能。

### ⚠ Warning

端點解析是進階 SDK 主題。如果您變更預設設定，則可能會破壞程式碼。預設設定適用於生產環境中的大多數使用者。

自訂端點可以全域設定，以使用於所有服務請求，或者您可以為特定設定自訂端點 AWS 服務。

您可以使用環境變數或共用 AWS config 檔案中的設定來設定自訂端點。如需此方法的資訊，請參閱 AWS SDKs 和工具參考指南中的[服務特定端點](#)。如需所有共用 config 檔案設定和環境變數的完整清單 AWS 服務，請參閱[服務特定端點的識別符](#)。

或者，您也可以可以在程式碼中設定此自訂，如下列各節所示。

## 自訂組態

您可以使用建置用戶端時可用的兩種方法來自訂服務用戶端的端點解析：

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

您可以設定這兩個屬性。不過，您通常只提供一個。對於一般用途，`endpoint_url` 最常自訂。

## 設定端點 URL

您可以為設定值 `endpoint_url`，以指出服務的「基礎」主機名稱。不過，此值不是最終值，因為它會以參數形式傳遞給用戶端的 `ResolveEndpoint` 執行個體。實作接著 `ResolveEndpoint` 可以檢查並可能修改該值，以判斷最終端點。

## 設定端點解析程式

服務用戶端的 `ResolveEndpoint` 實作會決定 SDK 用於任何指定請求的最終解析端點。服務用戶端會針對每個請求呼叫 `resolve_endpoint` 方法，並使用解析程式傳回 [EndpointFuture](#) 的值，而不會進一步變更。

下列範例示範為 Amazon S3 用戶端提供自訂端點解析程式實作，以解析每個階段的不同端點，例如預備和生產：

```

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

端點解析程式和ResolveEndpoint延伸特徵是每個服務特有的，因此只能在服務用戶端組態上設定。另一方面，端點 URL 可以使用共用組態（套用至衍生自它的所有服務）或特定服務進行設定。

## ResolveEndpoint 參數

此resolve\_endpoint方法接受服務特定的參數，其中包含端點解析中使用的屬性。

每個服務都包含下列基本屬性：

名稱	Type	描述
region	字串	用戶端的 AWS 區域

名稱	Type	描述
endpoint	字串	的值集的字串表示法 <code>endpointUrl</code>
use_fips	Boolean	用戶端組態中是否啟用 FIPS 端點
use_dual_stack	Boolean	是否在用戶端的組態中啟用雙堆疊端點

AWS 服務 可以指定解析度所需的其他屬性。例如，Amazon S3 [端點參數](#) 包含儲存貯體名稱和數個 Amazon S3-specific 功能設定。例如，`force_path_style` 屬性會判斷是否可以使用虛擬主機定址。

如果您實作自己的提供者，則不需要建構自己的端點參數執行個體。軟體開發套件為每個請求提供屬性，並將它們傳遞給您的實作 `resolve_endpoint`。

## 比較使用 `endpoint_url` 與使用 `endpoint_resolver`

請務必了解，以下兩種組態，一種是使用 `endpoint_url`，另一種是使用 `endpoint_resolver`，請勿產生具有同等端點解析行為的用戶端。

```
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();
```

設定的用戶端會 `endpoint_url` 指定傳遞至（預設）提供者的基本 URL，此 URL 可以修改為端點解析的一部分。

設定的用戶端會 `endpoint_resolver` 指定 Amazon S3 用戶端使用的最終 URL。

## 範例

自訂端點通常用於測試。呼叫會路由到本機託管的模擬服務，而不是呼叫雲端服務。兩個這類選項為：

- [DynamoDB local](#) – Amazon DynamoDB 服務的本機版本。
- [LocalStack](#) – 在本機機器的容器中執行的雲端服務模擬器。

下列範例說明指定自訂端點以使用這兩個測試選項的兩種不同方式。

### 直接在程式碼中使用 DynamoDB 本機

如前幾節所述，您可以 `endpoint_url` 直接在程式碼中設定，以覆寫基本端點，指向本機 DynamoDB 伺服器。在您的程式碼中：

```
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);
```

[完整的範例](#) 可在 GitHub 上取得。

### 使用 `config` 檔案使用 LocalStack

您可以在共用 AWS config 檔案中設定 [服務特定的端點](#)。下列組態設定檔集 `endpoint_url` 可在連接埠 `localhost` 上連線至 4566。如需 LocalStack 組態的詳細資訊，請參閱 [LocalStack 文件網站上的透過端點 URL 存取 LocalStack](#)。

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

軟體開發套件會擷取共用config檔案中的變更，並在您使用localstack設定檔時將其套用至軟體開發套件用戶端。使用此方法，您的程式碼不需要包含任何端點的參考，而且看起來像：

```
// 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);
```

[完整的範例](#)可在 GitHub 上取得。

## HTTP

適用於 Rust 的 AWS SDK 提供下列 HTTP 相關組態：

主題

- [Hyper 1.x](#)

### Hyper 1.x

#### Note

適用於 Rust 的 AWS SDK 是在hyper發行穩定 1.0 版本之前開發的。軟體開發套件目前預設為使用 v0.14 版本的 `hyper`

hyper 是 Rust 的較低層級 HTTP 程式庫，可與 搭配使用 適用於 Rust 的 AWS SDK ，以發出 API 服務呼叫。如需此條板箱的參考文件，請參閱 [Hyper - Rust](#)。根據預設，適用於 Rust 的 SDK 會使用 v0.14 版本 hyper和 CryptoMode:Ring提供者。



支援 hyper 1.0 的用戶端位於 [aws-smithy-experimental](#) 條板箱中。

## 選擇加密提供者

您必須選擇 `CryptoProvider` 並在 `aws-smithy-experimental` 條板箱中啟用所需的功能。

下列 `CryptoProvider` 模式可供使用：

- `CryptoMode:AwsLc` - 以 `aws-lc-rs` 為基礎的密碼編譯提供者 [aws-lc-rs](#)。這是由 AWS 密碼編譯團隊為 AWS 及其客戶維護的通用密碼編譯程式庫，以 Google BoringSSL 專案和 OpenSSL 專案的程式碼為基礎。
  - 需要功能 `crypto-aws-lc`。
- `CryptoMode:AwsLcFips` - 以 `aws-lc-rs` 為基礎的 FIPS 相容密碼編譯提供者 [aws-lc-rs](#)。
  - 需要功能 `crypto-aws-lc-fips`。

### Note

這可能需要額外的建置工具才能編譯。如需詳細資訊，請參閱 [aws-lc-rs](#) 儲存庫和建置說明。

- `CryptoMode:Ring` - 以 `ring` 為基礎的密碼編譯提供者 [ring](#)。這是「使用 Rust 的安全、快速、小型加密」。
  - 需要功能 `crypto-ring`。

### Note

這是 Rust 開發套件在預設 hyper v0.14 用戶端中使用的加密提供者。

## 如何啟用條板箱的功能

下列範例示範如何將 `crypto-aws-lc` 功能新增至專案中的 `aws-smithy-experimental` 條板箱 `Cargo.toml`：

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

或者，您可以在專案資料夾中的命令提示字元執行下列命令：

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

如需詳細資訊，請參閱 貨運手冊中的[功能](#)。

## 建立 hyper 1.0 用戶端並搭配 SDK 使用

若要建立 hyper 1.0 用戶端，請在專案的 Cargo.toml 新增 aws-smithy-experimental 條板箱。HyperClientBuilder 需要您選擇 CryptoProvider。上一節涵蓋提供者選擇，以及如何啟用 aws-smithy-experimental 條板箱的其中一個加密相關功能。

下列程式碼使用 CryptoMode::AwsLc 做為 hyper 1.0 HTTP 用戶端的加密提供者，以搭配 Amazon Simple Storage Service (Amazon S3) 服務用戶端使用。在此範例中，您專案的 Cargo.toml 也需要包含 AWS 服務的條板箱 aws-sdk-s3。

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

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

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

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

HTTP 用戶端可以在共用組態結構或特定服務的組態結構上設定。

## 攔截器

您可以使用攔截器來關聯至 API 請求和回應的執行。攔截器是開放式機制，其中 SDK 呼叫您編寫的程式碼，將行為注入請求/回應生命週期。如此一來，您就可以修改處理中的請求、偵錯請求處理、檢視錯誤等。

下列範例顯示一個簡單的攔截器，在輸入重試迴圈之前，會先將額外的標頭新增至所有傳出請求：

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

如需詳細資訊和可用的攔截器掛鉤，請參閱[攔截](#)特徵。

## 攔截器註冊

您在建構服務用戶端或覆寫特定操作的組態時註冊攔截器。根據您希望攔截器套用至用戶端的所有操作，還是只套用特定操作，註冊方式會有所不同。

### 服務用戶端上所有操作的攔截器

若要為整個用戶端註冊攔截器，請使用 `Builder` 模式新增攔截器。

```
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);
```

### 僅特定操作的攔截器

若要僅註冊單一操作的攔截器，請使用 `customize` 延伸模組。您可以使用此方法覆寫每個操作層級的服務用戶端組態。如需可自訂操作的詳細資訊，請參閱[覆寫用戶端的單一操作組態](#)。

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

## 使用可觀測性功能

可觀測性是可從系統發出的資料推斷系統目前狀態的程度。發出的資料通常稱為遙測。

### 主題

- [啟用適用於 Rust 的 AWS SDK 程式碼記錄](#)

## 啟用 適用於 Rust 的 AWS SDK 程式碼記錄

適用於 Rust 的 AWS SDK 使用[追蹤](#)架構進行記錄。

1. 在專案目錄的命令提示中，新增[追蹤訂閱者](#)目錄做為相依性：

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

這會將 條板箱新增至 Cargo.toml 檔案的 [dependencies] 區段。

2. 初始化訂閱者。通常在 main 函式的早期完成，然後再呼叫任何適用於 Rust 的 SDK 操作：

```
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 環境變數開啟記錄。若要啟用記錄資訊的顯示，請在命令提示中，將 RUST\_LOG 環境變數設定為您要記錄的層級。下列範例會將記錄設定為 debug 層級：

Linux/macOS

```
$ RUST_LOG=debug
```

## Windows

如果您使用的是 VSCode，終端機視窗通常會預設為 PowerShell。驗證您正在使用的提示類型。

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

## PowerShell

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

### 4. 執行程式：

```
$ cargo run
```

您應該會在主控台或終端機視窗中看到其他輸出。

如需詳細資訊，請參閱從 `tracing-subscriber` 文件[篩選具有環境變數的事件](#)。

## 解譯日誌輸出

按照上一節中的步驟開啟日誌記錄後，預設會將其他日誌資訊列印為標準輸出。

如果您使用的是預設日誌輸出格式（追蹤模組稱為「完整」），您在日誌輸出中看到的資訊如下所示：

```
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
```

每個項目都包含下列項目：

- 日誌項目的時間戳記。
- 項目的日誌層級。這是 INFO、DEBUG 或 等單字 TRACE。
- 產生日誌項目的巢狀範圍集，以冒號 (":") 分隔。這可協助您識別日誌項目的來源。
- Rust 模組路徑，其中包含產生日誌項目的程式碼。
- 記錄訊息文字。

追蹤模組的標準輸出格式使用 ANSI 逸出碼來將輸出著色。篩選或搜尋輸出時，請記住這些逸出序列。

#### Note

在巢狀範圍集中 `sdk_invocation_id` 顯示的是 SDK 產生的唯一 ID 用戶端，可協助建立日誌訊息的關聯。它與 回應中找到的請求 ID 無關 AWS 服務。

## 微調您的記錄層級

如果您使用支援環境篩選的 條板箱，例如 `tracing_subscriber`，您可以依模組控制日誌的真實性。

您可以為每個模組開啟相同的記錄層級。下列會將 `trace` 每個模組的記錄層級設定為：

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

您可以開啟特定模組的追蹤層級記錄。在下列範例中，只有來自的日誌 `aws_smithy_runtime` 會進入 `trace` 關卡。

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

您可以為多個模組指定不同的日誌層級，方法是使用逗號分隔它們。下列範例會將 `aws_config` 模組設定為 `trace` 關卡記錄，並將 `aws_smithy_runtime` 模組設定為 `debug` 關卡記錄。

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

下表說明您可以用來篩選日誌訊息的一些模組：

字首	描述
aws_smithy_runtime	請求和回應線路記錄
aws_config	登入資料載入
aws_sigv4	請求簽署和正式請求

找出您需要在日誌輸出中包含哪些模組的一種方法是先記錄所有項目，然後在日誌輸出中尋找所需資訊的條板箱名稱。然後，您可以相應地設定環境變數，然後再次執行您的程式。

## 覆寫用戶端的單一操作組態

[建立服務用戶端](#)之後，組態會變成不可變，並套用至所有後續操作。雖然目前無法修改組態，但可以依每次操作覆寫組態。

每個操作建置器都有可用來建立 customize 的方法，CustomizableOperation 讓您可以覆寫現有組態的個別複本。原始用戶端組態將保持不變。

下列範例顯示建立呼叫兩個操作的 Amazon S3 用戶端，其中第二個會覆寫以傳送至不同的 AWS 區域。所有 Amazon S3 的物件調用都會使用 us-east-1 區域，但明確覆寫 API 呼叫以使用修改的時除外 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

先前的範例適用於 Amazon S3，但所有操作的概念都相同。某些操作在上可能會有其他方法 `CustomizableOperation`。

如需使用 `customize` 為單一操作新增攔截器的範例，請參閱 [僅特定操作的攔截器](#)。

## 重試

適用於 Rust 的 AWS SDK 提供預設重試行為和可自訂的組態選項。呼叫 AWS 服務偶爾會傳回非預期的例外狀況。如果重試呼叫，某些類型的錯誤，例如限流或暫時性錯誤可能會成功。

重試行為可以使用共用 AWS config 檔案中的環境變數或設定進行全域設定。如需此方法的資訊，請參閱 AWS SDKs 和工具參考指南中的 [重試行為](#)。它也包含重試策略實作的詳細資訊，以及如何選擇其中一個。

或者，您也可以程式碼中設定這些選項，如以下各節所示。

### 預設重試組態

每個服務用戶端都會預設為透過 [RetryConfig](#) 結構提供的 `standard` 重試策略組態。根據預設，會嘗試三次呼叫（初次嘗試，加上兩次重試）。此外，每次重試都會延遲一小段時間，以避免重試風暴。此慣例適用於大多數使用案例，但在高輸送量系統等特定情況下可能不合適。

只有某些類型的錯誤會被 SDKs 視為可重試。可重試錯誤的範例包括：

- 通訊端逾時

- 服務端調節
- HTTP 5XX 回應等暫時性服務錯誤

下列範例不視為可重試：

- 缺少參數或其無效
- 驗證/安全錯誤
- 設定錯誤例外狀況

您可以透過設定最大嘗試次數、延遲和退避組態來自訂 standard 重試策略。

## 最大嘗試次數

您可以提供修改過的 [RetryConfig](#) 給 `aws_config::defaults`，以自訂程式碼中的嘗試次數上限 `aws_config::defaults`：

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

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

## 延遲和退避

如果需要重試，預設重試策略會先等待，再進行後續嘗試。第一次重試的延遲很小，但之後重試會呈指數增長。延遲量上限會設定為不會太大。

隨機抖動會套用至所有嘗試之間的延遲。抖動有助於減輕大型機群可能造成重試風暴的影響。如需指數退避和抖動的深入討論，請參閱 AWS 架構部落格中的 [指數退避和抖動](#)。

您可以提供修改過的 [RetryConfig](#) 給 `aws_config::defaults`，以自訂程式碼中的延遲設定 `aws_config::defaults`。下列程式碼會將組態設定為將第一次重試嘗試延遲最多 100 毫秒，且任何重試嘗試之間的時間上限為 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;
```

## 自適應重試模式

作為standard模式重試策略的替代方案，adaptive模式重試策略是一種進階方法，可尋找理想的請求速率，以將限流錯誤降至最低。

### Note

自適應重試是進階重試模式。通常不建議使用此策略。請參閱 AWS SDKs和工具參考指南中的[重試行為](#)。

自適應重試包含標準重試的所有功能。它新增了用戶端速率限制器，可測量節流請求的速率，而不是非節流請求。它也會限制流量以嘗試保持在安全頻寬內，理想情況下會導致零限流錯誤。

速率會即時適應不斷變化的服務條件和流量模式，並可能相應地增加或減少流量速率。最重要的是，速率限制器可能會延遲高流量情況下的初始嘗試。

您可以透過提供修改過的 `RetryConfig`，在程式碼中選取adaptive重試策略[RetryConfig](#)：

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

## 逾時

適用於 Rust 的 AWS SDK 提供數種用於管理請求逾時和停滯資料串流的設定。這些有助於您的應用程式在網路中發生意外的延遲和故障時，以最佳方式運作。

## API 逾時

當有暫時性問題可能導致請求嘗試需要很長的時間或完全失敗時，請務必檢閱和設定逾時，以便您的應用程式可以快速失敗，並採取最佳行為。開發套件可以自動重試失敗的請求。設定個別嘗試和整個請求的逾時是很好的做法。

適用於 Rust 的 SDK 提供預設逾時，用於建立請求的連線。軟體開發套件沒有為接收請求嘗試或整個請求的回應設定任何預設最長等待時間。可使用下列逾時選項：

參數	預設值	描述
連線逾時	3.1 秒	放棄之前等待建立連線的時間上限。
操作逾時	無	從適用於 Rust 的 SDK 接收回應前的等待時間上限，包括所有重試。
操作嘗試逾時	無	等待單一 HTTP 嘗試的時間上限，之後可以重試 API 呼叫。
讀取逾時	無	從啟動請求時開始，等待讀取回應第一個位元組的時間上限。

下列範例顯示具有自訂逾時值的 Amazon S3 用戶端組態：

```
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);
```

當您同時使用 操作和嘗試逾時時，您可以針對所有重試嘗試所花費的總時間設定硬性限制。您也可以將個別 HTTP 請求設定為在慢速請求上快速失敗。

除了在所有操作的服務用戶端上設定這些逾時值之外，您也可以[針對單一請求 設定或覆寫這些值](#)。

### ⚠ Important

操作和嘗試逾時不適用於 Rust 開發套件傳回回應之後使用的串流資料。例如，從回應 `ByteStream` 的成員取用資料不受操作逾時限制。

## 停滯的串流保護

適用於 Rust 的 SDK 提供另一種與偵測停滯串流相關的逾時形式。停滯的串流是上傳或下載串流，不會產生超過設定寬限期的資料。這有助於防止應用程式無限期地懸置，且永遠不會進行進度。

當串流閒置的時間超過可接受的期間時，停滯的串流保護將傳回錯誤。

依預設，適用於 Rust 的 SDK 會同時啟用上傳和下載的停滯串流保護，並尋找至少 1 位元組/秒的活動，且具有 20 秒的寬廣寬限期。

下列範例顯示自訂的 `StalledStreamProtectionConfig`，可停用上傳保護並將沒有活動的寬限期變更為 10 秒：

```
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

停滯的串流保護是進階組態選項。建議您只在應用程式需要更緊密的效能，或導致其他問題時，才變更這些值。

## 停用停滯串流保護

下列範例示範如何完全停用停滯的串流保護：

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

# 使用 適用於 Rust 的 AWS SDK

了解使用 適用於 Rust 的 AWS SDK 來使用 AWS 服務的常見和建議方法。

## 主題

- [提出請求](#)
- [最佳實務](#)
- [並行數量](#)
- [錯誤處理](#)
- [建立 Lambda 函數](#)
- [分頁](#)
- [建立預先簽章URLs](#)
- [單元測試](#)
- [等待程式](#)

## 提出請求

若要向 提出請求 AWS 服務，您必須先[建立服務用戶端](#)。對於 AWS 服務 每個程式碼使用，它都有自己的條板箱和自己的 Client 來與其互動。

會針對服務公開的每個 API 操作Client公開一種方法。這些方法的傳回值是「流動建置器」，其中建置器樣式函數呼叫鏈結會新增該 API 的不同輸入。呼叫服務的方法後，請呼叫 `send()` 以取得 [Future](#) 輸出成功或的 `SdkError`。如需 `SdkError` 的詳細資訊，請參閱[錯誤處理](#)。

下列範例示範使用 Amazon S3 在 中建立儲存貯體的基本操作us-west-2 AWS 區域：

```
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;
```

每個服務箱都有用於 API 輸入的其他模組，例如：

- `types` 模組具有結構或列舉，可提供更複雜的結構化資訊。
- `primitives` 模組具有更簡單的類型，可代表日期時間或二進位 Blob 等資料。

如需更詳細的條板箱組織和資訊，請參閱服務條板箱的 [API 參考文件](#)。例如，Amazon Simple Storage Service 的 `aws-sdk-s3` 條板箱有數個 [模組](#)。其中兩個是：

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

## 最佳實務

以下是使用的最佳實務 適用於 Rust 的 AWS SDK。

### 盡可能重複使用 SDK 用戶端

根據 SDK 用戶端的建構方式，建立新的用戶端可能會導致每個用戶端維護自己的 HTTP 連線集區、身分快取等。我們建議共用用戶端或至少共用 `SdkConfig`，以避免昂貴資源建立的額外負荷。所有 SDK 用戶端都會實作 `Clone` 為單一原子參考計數更新。

### 設定 API 逾時

軟體開發套件提供一些逾時選項的預設值，例如連線逾時和通訊端逾時，但不適用於 API 呼叫逾時或個別 API 呼叫嘗試。最好同時為個別嘗試和整個請求設定逾時。這將確保您的應用程式在發生暫時性問題時以最佳方式快速失敗，而這些問題可能導致請求嘗試花費更長的時間來完成或導致死亡的網路問題。

如需設定操作逾時的詳細資訊，請參閱 [逾時](#)。



## 並行數量

適用於 Rust 的 AWS SDK 不提供並行控制，但使用者有許多實作自己的選項。

### 條款

與此主題相關的術語很容易混淆，有些術語已經成為同義詞，即使它們最初代表不同的概念。在本指南中，我們將定義以下內容：

- 任務：您的程式將執行到完成或嘗試執行到完成的一些「工作單位」。
- 循序運算：依序執行多個任務時。
- 並行運算：在重疊時段執行多個任務時。
- 並行：電腦以任意順序完成多個任務的能力。
- 多工處理：電腦同時執行多個任務的能力。
- 競賽條件：程式的行為會根據任務啟動的時間或處理任務所需的時間而變更。
- 爭用：對共用資源的存取發生衝突。當兩個或多個任務想要同時存取資源時，該資源是「爭用中」。
- Deadlock：無法再進行進度的狀態。這通常是因為兩個任務想要取得彼此的資源，但兩個任務都不會釋出其資源，直到對方的資源可用為止。死鎖會導致程式變得部分或完全沒有回應。

### 簡單範例

我們的第一個範例是循序程式。在後續範例中，我們將使用並行技術來變更此程式碼。稍後的範例會重複使用相同的 `build_client_and_list_objects_to_download()` 方法，並在 `main()` 中進行變更。

下列範例任務是下載 Amazon Simple Storage Service 儲存貯體中的所有檔案：

1. 首先列出所有檔案。將金鑰儲存在清單中。
2. 反覆查看清單，依序下載每個檔案

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

// This initialization function won't be reproduced in
// examples following this one, in order to save space.
async fn build_client_and_list_objects_to_download() {
    let cfg = aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
    let client = Client::new(&cfg);
```

```
let objects_to_download: Vec<_> = client
    .list_objects_v2()
    .bucket(EXAMPLE_BUCKET)
    .send()
    .await
    .expect("listing objects succeeds")
    .contents()
    .into_iter()
    .flat_map(aws_sdk_s3::types::Object::key)
    .map(ToString::to_string)
    .collect();

(client, objects_to_download)
}
```

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

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

### Note

在這些範例中，我們不會處理錯誤，而且假設範例儲存貯體沒有具有看起來像檔案路徑之金鑰的物件。因此，我們不會涵蓋建立巢狀目錄。

由於現代電腦的架構，我們可以重寫此程式，以提高效率。我們會在稍後的範例中執行此作業，但首先，我們再學習幾個概念。

## 擁有權和可變性

Rust 中的每個值都有一個擁有者。當擁有者超出範圍時，也會捨棄其擁有的所有值。擁有者可以提供一或多個不可變的值參考或單一可變參考。Rust 編譯器負責確保沒有任何參考過期其擁有者。

當多個任務需要可變存取相同的資源時，需要額外的規劃和設計。在循序運算中，每個任務都可以在不爭用的情況下可變存取相同的資源，因為它們會依序執行。不過，在並行運算中，任務可以按任何順序同時執行。因此，我們必須做更多事向編譯器證明無法進行多個可變參考（或至少在發生時當機）。

Rust 標準程式庫提供許多工具來協助我們達成此目標。如需這些主題的詳細資訊，請參閱 [The Rust Programming Language Book](#) 中的 [變數和可變動性和了解所有權](#)。

### 更多詞彙！

以下是「同步物件」的清單。總而言之，它們是說服編譯器所需的工具，我們的並程式不會破壞所有權規則。

#### [標準程式庫同步物件](#)：

- [Arc](#)：A 斷層 R eference-C 引號。當資料在 中包裝時Arc，可以自由共用，而無需擔心任何特定擁有者提早捨棄該值。在這種情況下，值的擁有權會變成「共用」。內的值Arc不能可變，但可能具有[內部可變性](#)。
- [障礙](#)：確保多個執行緒會等待彼此到達程式中的某個點，然後再繼續一起執行。
- [Condvar](#)：Cond ition Var iable，可讓您在等待事件發生時封鎖執行緒。
- [Mutex](#)：一種 Mut ual Ex clusion 機制，可確保一次最多有一個執行緒能夠存取一些資料。一般而言，絕不應將Mutex鎖定保留在程式碼中的某個.await點。

#### [Tokio 同步物件](#)：

雖然 AWS SDKs 旨在與async執行期無關，但我們建議在特定情況下使用tokio同步物件。

- [Mutex](#)：類似於標準程式庫的 Mutex，但成本略高。與標準不同Mutex，這個項目可以保留在程式碼中的某個.await點。
- [Semaphore](#)：用於控制多個任務對常見資源之存取的變數。

## 重寫我們的範例，以提升效率（單執行緒並行）

在下列修改範例中，我們使用 [futures\\_util::future::join\\_all](#) 來同時執行所有 `get_object` 請求。

```
#[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;
}
```

這是受益於並行的最簡單方法，但它也有幾個問題，一開始可能並不明顯：

1. 我們會同時建立所有請求輸入。如果記憶體不足，無法保留所有 `get_object` 請求輸入，則會遇到「out-of-memory」配置錯誤。
2. 我們同時建立和等待所有未來。如果我們嘗試一次下載過多，Amazon S3 會調節請求。

若要修正這兩個問題，我們必須限制我們一次傳送的請求數量。我們會使用 `tokio` [旗號](#) 來執行此操作：

```
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;
}
```

我們已將請求建立移至 `async` 區塊，以修正潛在的記憶體用量問題。如此一來，在傳送請求之前，系統不會建立請求。

#### Note

如果您有記憶體，則一次建立所有請求輸入並將其保留在記憶體中，直到準備好傳送為止，可能會更有效率。若要嘗試這麼做，請將請求輸入建立移至 `async` 區塊之外。

我們也已修正一次傳送過多請求的問題，方法是限制傳輸中的請求至 `CONCURRENCY_LIMIT`。

### Note

的正確值會因每個專案 `CONCURRENCY_LIMIT` 而異。建構和傳送自己的請求時，請嘗試盡可能將其設定為最高，而不會執行調節錯誤。雖然可以根據服務傳回的成功限流回應比率動態更新並行限制，但由於其複雜性，因此超出本指南的範圍。

## 重寫我們的範例以提高效率（多執行緒並行）

在前面兩個範例中，我們同時執行了請求。雖然這比同步執行它們更有效率，但我們可以透過使用多執行緒讓事情更有效率。若要使用執行此作業 `tokio`，我們需要將它們產生為個別任務。

### Note

此範例要求您使用多執行緒 `tokio` 執行時間。此執行時間位於 `rt-multi-thread` 功能後方。當然，您需要在多核心機器上執行程式。

```
// 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;
}
```

將工作分成任務可能很複雜。執行 I/O (輸入/輸出) 通常會遭到封鎖。執行期可能會難以平衡長時間執行任務與短期執行任務的需求。無論您選擇哪個執行時間，請務必閱讀他們的建議，以最有效率的方式將工作分成任務。如需 tokio 執行時間建議，請參閱[模組 `tokio::task`](#)。

## 偵錯多執行緒應用程式

可依任何順序同時執行的任務。因此，並行程式的日誌可能很難讀取。在適用於 Rust 的 SDK 中，我們建議您使用 `tracing` 記錄系統。無論日誌何時執行，它都可以將日誌與其特定任務分組。如需準則，請參閱[啟用 適用於 Rust 的 AWS SDK 程式碼記錄](#)。

識別鎖定任務的實用工具是 [tokio-console](#)，這是非同步 Rust 程式的診斷和偵錯工具。透過檢測和執行您的程式，然後執行 `tokio-console` 應用程式，您可以看到程式正在執行的任務的即時檢視。此檢視包含有用的資訊，例如任務等待取得共用資源所花費的時間，或是輪詢資源的時間。

## 錯誤處理

了解 適用於 Rust 的 AWS SDK 傳回錯誤的方式和時間對於使用 SDK 建置高品質應用程式至關重要。下列各節說明您可能從 SDK 遇到的不同錯誤，以及如何適當處理這些錯誤。

每個操作都會傳回錯誤類型設為的 `Result` 類型 [`SdkError<E, R = HttpResponse>`](#)。`SdkError` 是具有多種可能類型的列舉，稱為變體。

## 服務錯誤

最常見的錯誤類型是 `SdkError::ServiceError`。此錯誤代表來自的錯誤回應 AWS 服務。例如，如果您嘗試從 Amazon S3 取得不存在的物件，Amazon S3 會傳回錯誤回應。

當您遇到 `SdkError::ServiceError`，表示您的請求已成功傳送至，AWS 服務但無法處理。這可能是因為請求參數中的錯誤，或因為服務端的問題。

錯誤回應詳細資訊包含在錯誤變體中。下列範例示範如何方便地取得基礎 `ServiceError` 變體並處理不同的錯誤案例：

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

## 錯誤中繼資料

每個服務錯誤都有額外的中繼資料，可透過匯入服務特定的特徵來存取。

- `<service>::error::ProvideErrorMetadata` 特徵可讓您存取任何可用的基礎原始錯誤碼，以及從服務傳回的錯誤訊息。



- 對於 Amazon S3，此特徵為 [aws\\_sdk\\_s3::error::ProvideErrorMetadata](#)。

您也可以取得在疑難排解服務錯誤時可能有用的資訊：

- `<service>::operation::RequestId` 特徵會新增延伸方法，以擷取服務產生的唯一 AWS 請求 ID。
  - 對於 Amazon S3，此特徵為 [aws\\_sdk\\_s3::operation::RequestId](#)。
- `<service>::operation::RequestIdExt` 特徵會新增 `extended_request_id()` 方法，以取得額外的延伸請求 ID。
  - 僅部分 服務支援。
  - 對於 Amazon S3，此特徵為 [aws\\_sdk\\_s3::operation::RequestIdExt](#)。

## 使用 列印的詳細錯誤 `DisplayErrorContext`

開發套件中的錯誤通常是失敗鏈的結果，例如：

1. 因為連接器傳回錯誤，所以傳送請求失敗。
2. 連接器傳回錯誤，因為登入資料提供者傳回錯誤。
3. 登入資料提供者傳回錯誤，因為它稱為 服務，且該服務傳回錯誤。
4. 服務傳回錯誤，因為登入資料請求沒有正確的授權。

根據預設，此錯誤的顯示只會輸出「分派失敗」。這缺少有助於對錯誤進行故障診斷的詳細資訊。適用於 Rust 的 SDK 提供名為 `DisplayErrorContext` 的簡單錯誤報告程式。

- `<service>::error::DisplayErrorContext` 結構新增了輸出完整錯誤內容的功能。
  - 對於 Amazon S3，此結構為 [aws\\_sdk\\_s3::error::DisplayErrorContext](#)。

包裝要顯示並列印的錯誤時，`DisplayErrorContext` 會提供更詳細的訊息，如下所示：

```
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") },
              { _private: H0("114") },
              "content-length": HeaderValue
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  
}  
}  
)  
}
```

## 建立 Lambda 函數

如需使用 開發 AWS Lambda 函數的詳細文件 適用於 Rust 的 AWS SDK，請參閱《AWS Lambda 開發人員指南》中的[使用 Rust 建置 Lambda 函數](#)。該文件會引導您使用：

- 核心功能的 Rust Lambda 執行期用戶端目錄，[aws-lambda-rust-runtime](#)。
- 使用 Cargo Lambda 將 Rust 函數二進位檔部署至 [Lambda](#) 的建議命令列工具。

除了 AWS Lambda 開發人員指南中的引導式範例之外，GitHub 上的 [AWS SDK 程式碼範例儲存庫](#) 也提供 Lambda 計算器範例。

## 分頁

當承載太大而無法在單一回應中傳回時，許多 AWS 操作會傳回截斷的結果。反之，服務會傳回一部分的資料和權杖，以擷取下一組項目。此模式稱為分頁。

適用於 Rust 的 AWS SDK 包含操作建置器 `into_paginator` 上的延伸方法，可用於自動為您分頁結果。您只需撰寫處理結果的程式碼。所有分頁操作建置器都有可用的 `into_paginator()` 方法，可讓 `PaginationStream<Item>` 分頁結果。

- 在 Amazon S3 中，其中一個範例是

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

下列範例使用 Amazon Simple Storage Service。不過，對於具有一或多個分頁 APIs 的任何服務，這些概念都相同。

下列程式碼範例顯示最簡單的範例，使用 `try_collect()` 方法將所有分頁結果收集到 Vec：

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

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

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

有時候，您希望對分頁有更多控制權，而不是一次將所有內容全部提取到記憶體中。下列範例會反覆運算 Amazon S3 儲存貯體中的物件，直到不再存在為止。

```
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);
    }
}
```

## 建立預先簽章URLs

您可以為某些 AWS API 操作預先簽署請求，以便其他發起人稍後可以使用請求，而無需出示自己的登入資料。

例如，假設 Jane 可以存取 Amazon Simple Storage Service (Amazon S3) 物件，而且她想要暫時與 Alejandro 共用物件存取。Jane 可以產生預先簽章的 GetObject 請求，以與 Alejandro 共用，因此他可以下載物件，而不需要存取 Jane 的登入資料或擁有自己的任何登入資料。預先簽章 URL 使用的登入資料是 Jane 的，因為她是產生 URL AWS 的使用者。

若要進一步了解 Amazon S3 中預先簽章URLs，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用預先簽章URLs](#)。

## 預先簽章基本概念

適用於 Rust 的 AWS SDK 提供操作流暢建置器 `presigned()` 的方法，可用來取得預先簽章的請求。

下列範例會建立 Amazon S3 的預先簽章 GetObject 請求。請求在建立後 5 分鐘內有效。

```
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()` 方法會傳回 `Result<PresignedRequest, SdkError<E, R>>`。

傳回的 `PresignedRequest` 包含在 HTTP 請求元件取得的方法，包括方法、URI 和任何標頭。所有這些都需要傳送到服務，如果有的話，請求才會有效。不過，許多預先簽章的請求可以單獨由 URI 表示。

## 預先簽章POST和PUT請求

許多可預先簽章的操作只需要 URL，而且必須以 HTTP GET 請求傳送。不過，某些操作會採用內文，在某些情況下，必須以 HTTP POST 或 HTTP PUT 請求的形式與標頭一起傳送。預先簽署這些請求與預先簽署 GET 請求相同，但調用預先簽署的請求更為複雜。

以下是預先簽署 Amazon S3 `PutObject` 請求並將其轉換為的範例 [http::request::Request](#)，該請求可以使用您選擇的 HTTP 用戶端傳送。

若要使用 `into_http_1x_request()` 方法，請將 `http-1x` 功能新增至 `Cargo.toml` 檔案中的 `aws-sdk-s3` 條板箱：

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

來源檔案：

```
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);
```

## 獨立簽署者

### Note

這是進階使用案例。大多數使用者不需要或建議這樣做。

有幾個使用案例需要建立 SDK for Rust 內容之外的已簽署請求。對於 [aws-sigv4](#) 條板箱，您可以獨立於 SDK 使用 [aws-sigv4](#) 條板箱。

以下是示範基本元素的範例，如需詳細資訊，請參閱 [條板箱文件](#)。

將 `aws-sigv4` 和 `http` 條板箱新增至您的 `Cargo.toml` 檔案：

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

來源檔案：

```
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);
```

## 單元測試

雖然您可以在適用於 Rust 的 AWS SDK 專案中實作單元測試的方法有很多種，但我們建議您執行以下幾個操作：

- [automock](#) 使用 [mockall](#) 條板箱中的 來建立和執行您的測試。
- 使用 AWS Smithy 執行期的 [StaticReplayClient](#) 來建立可以用來取代標準 HTTP 用戶端的仿造 HTTP 用戶端 AWS 服務。此用戶端會傳回您指定的 HTTP 回應，而不是透過網路與服務通訊，以便測試取得已知資料以供測試之用。

## 使用模擬模型自動產生模擬

您可以使用 [automockmockall](#) 從條板箱中常用的，自動產生測試所需的大多數模擬實作。

此範例會測試名為的自訂方法 `determine_prefix_file_size()`。此方法會呼叫呼叫 Amazon S3 的自訂 `list_objects()` 包裝函式方法。透過模擬 `list_objects()`，可以測試 `determine_prefix_file_size()` 方法，而無需實際聯絡 Amazon S3。

1. 在專案目錄的命令提示中，新增 [mockall](#) 條板箱做為相依性：

```
$ cargo add mockall
```



這會將 條板箱新增至 Cargo.toml 檔案的 [dependencies] 區段。

2. 包含mockall來自條板箱的automock模組。

同時包含與您正在測試 AWS 服務 之 相關的任何其他程式庫，在此案例中為 Amazon S3。

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

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

3. 接下來，新增程式碼，以決定應用程式 Amazon S3 包裝函式結構的兩個實作中要使用哪個。
  - 寫來透過網路存取 Amazon S3 的真實內容。
  - 產生的模擬實作mockall。

在此範例中，所選取的名稱會命名為 S3。根據test屬性來選擇條件式：

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

4. S3Impl 結構是 Amazon S3 包裝函式結構的實作，其實際會將請求傳送至 AWS。
  - 啟用測試時，不會使用此程式碼，因為請求會傳送到模擬而非模擬 AWS。dead\_code 如果未使用 S3Impl 類型，屬性會告知 linter 不報告問題。
  - 條件式#[cfg\_attr(test, automock)]表示啟用測試時，應設定 automock 屬性。這會通知 mockall 產生將命名 S3Impl 為 的模擬 MockS3Impl。
  - 在此範例中，list\_objects() 方法是您想要模擬的呼叫。automock 會自動為您建立 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. 在名為 `test` 的模組中建立測試函數 `test`。

- 條件式 `#[cfg(test)]` 表示如果 `test` 屬性為 `true`，`mockall` 應該建置測試模組 `true`。

```

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

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

```

```
        ]))
        .build())
    });

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

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

#[tokio::test]
async fn test_multiple_pages() {
    // Create the Mock instance with two pages of objects now
    let mut mock = MockS3Impl::default();
    mock.expect_list_objects()
        .with(eq("test-bucket"), eq("test-prefix"), eq(None))
        .return_once(|_, _, _| {
            Ok(ListObjectsV2Output::builder()
                .set_contents(Some(vec![
                    // Mock content for ListObjectsV2 response
                    s3::types::Object::builder().size(5).build(),
                    s3::types::Object::builder().size(2).build(),
                ]))
                .set_next_continuation_token(Some("next".to_string()))
                .build())
        });
    mock.expect_list_objects()
        .with(
            eq("test-bucket"),
            eq("test-prefix"),
            eq(Some("next".to_string())),
        )
        .return_once(|_, _, _| {
            Ok(ListObjectsV2Output::builder()
                .set_contents(Some(vec![
                    // Mock content for ListObjectsV2 response
                    s3::types::Object::builder().size(3).build(),
                    s3::types::Object::builder().size(9).build(),
                ]))
                .build())
        });
};
```

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

    assert_eq!(19, size);
}
}
```

- 每個測試都會使用 `let mut mock = MockS3Impl::default();` 來建立 `mock` 執行個體 `MockS3Impl`。
  - 它會使用模擬的 `expect_list_objects()` 方法 (由自動建立 `automock`)，設定在程式碼中其他位置使用該 `list_objects()` 方法時的預期結果。
  - 建立預期之後，它會使用這些預期來呼叫來測試函數 `determine_prefix_file_size()`。使用聲明檢查傳回的值以確認其正確。
6. `determine_prefix_file_size()` 函數使用 Amazon S3 包裝函式來取得字首檔案的大小：

```
#[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)
}
```

類型 `S3` 用於呼叫 Rust 函數的包裝 SDK，以在提出 HTTP 請求 `MockS3Impl` 時同時支援 `S3Impl` 和。模擬會在啟用測試時 `mockall` 自動報告任何測試失敗。

您可以在 GitHub 上 [檢視這些範例的完整程式碼](#)。

## 使用靜態重播模擬 HTTP 流量

`aws-smithy-runtime` 條板箱包含名為 `StaticReplayClient` 的測試公用程式類別。您可以在建立 AWS 服務物件時指定此 HTTP 用戶端類別，而非預設的 HTTP 用戶端。

初始化時 `StaticReplayClient`，您會提供 HTTP 請求和回應的清單做為 `ReplayEvent` 物件。測試執行時，會記錄每個 HTTP 請求，而用戶端會傳回 `ReplayEvent` 事件清單中的下一個 HTTP 回應做為 HTTP 用戶端的回應。這可讓測試使用已知資料執行，無需網路連線。

### 使用靜態重播

若要使用靜態重播，您不需要使用包裝函式。反之，請判斷測試將使用的資料的實際網路流量應是什麼樣子，並在每次 SDK 從 AWS 服務用戶端發出請求時，提供該流量資料給 `StaticReplayClient` 使用。

#### Note

有數種方法可收集預期的網路流量，包括 AWS CLI 和許多網路流量分析器和封包偵測器工具。

- 建立 `ReplayEvent` 物件清單，指定預期的 HTTP 請求，以及應該傳回的回應。
- `StaticReplayClient` 使用上一個步驟中建立的 HTTP 交易清單來建立。
- 為 AWS 用戶端建立組態物件，將指定 `StaticReplayClient` 為 `Config` 物件的 `http_client`。
- 使用上一個步驟中建立的組態來建立 AWS 服務用戶端物件。
- 使用設定為使用的服務物件，執行您要測試的操作 `StaticReplayClient`。每次 SDK 傳送 API 請求到時 AWS，都會使用清單中的下一個回應。

**Note**

即使傳送的請求與ReplayEvent物件向量中的回應不相符，一律會傳回清單中的下一個回應。

- 完成所有所需的請求後，請呼叫 `StaticReplayClient.assert_requests_match()` 函數，確認 SDK 傳送的請求符合ReplayEvent物件清單中的請求。

## 範例

讓我們看看上一個範例中相同 `determine_prefix_file_size()` 函數的測試，但使用靜態重播而非模擬。

1. 在專案目錄的命令提示中，新增 [aws-smithy-runtime](#) 條板箱做為相依性：

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

這會將 條板箱新增至 `Cargo.toml` 檔案的 `[dependencies]` 區段。

2. 在來源檔案中，包含您需要的 `aws_smithy_runtime` 類型。

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

3. 測試一開始會建立ReplayEvent結構，代表測試期間應進行的每個 HTTP 交易。每個事件都包含 HTTP 請求物件和 HTTP 回應物件，代表 AWS 服務 通常會回覆的資訊。這些事件會傳遞至 的呼叫 `StaticReplayClient::new()`：

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

```

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

```

結果會存放在 `replay_client` 中。這表示 HTTP 用戶端，之後可在用戶端的組態中指定 Rust 的 SDK 來使用。

- 若要建立 Amazon S3 用戶端，請呼叫用戶端類別的 `from_conf()` 函數，使用組態物件建立用戶端：

```

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(),
);

```

組態物件是使用建置器的 `http_client()` 方法指定，而登入資料是使用 `credentials_provider()` 方法指定。登入資料是使用名為 `make_s3_test_credentials()` 的函數建立，該函數會傳回假的登入資料結構：

```

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

```

```
)  
}
```

這些登入資料不需要有效，因為它們實際上不會傳送到 AWS。

5. 呼叫需要測試的函數來執行測試。在此範例中，該函數的名稱為 `determine_prefix_file_size()`。其第一個參數是用於其請求的 Amazon S3 用戶端物件。因此，請指定使用建立的用戶端，`StaticReplayClient` 以便由該用戶端處理請求，而不是透過網路傳出：

```
let size = determine_prefix_file_size(client, "test-bucket", "test-prefix")  
    .await  
    .unwrap();  
  
assert_eq!(19, size);  
  
replay_client.assert_requests_match(&[]);
```

完成對的呼叫後 `determine_prefix_file_size()`，會使用宣告來確認傳回的值符合預期值。然後，會呼叫 `StaticReplayClient` 方法 `assert_requests_match()` 函數。此函數會掃描記錄的 HTTP 請求，並確認它們都符合建立重播用戶端時所提供 `ReplayEvent` 物件陣列中指定的請求。

您可以在 GitHub 上 [檢視這些範例的完整程式碼](#)。

## 等待程式

等待者是用戶端抽象，用來輪詢資源，直到達到所需的狀態，或直到確定資源不會進入所需的狀態為止。這是在使用最終一致服務時常見的任務，例如 Amazon Simple Storage Service，或是非同步建立資源的服務，例如 Amazon Elastic Compute Cloud。編寫邏輯以持續輪詢資源的狀態可能會很麻煩且容易出錯。等待者的目標是將此責任移出客戶程式碼，並移入適用於 Rust 的 AWS SDK，而對 AWS 操作的時間層面有深入的了解。

AWS 服務 為等待者提供支援，包括 `<service>::waiters` 模組。

- `<service>::client::Waiters` 特徵提供用戶端的等待程式方法。方法會針對 Client 結構實作。所有等待程式方法都遵循的標準命名慣例 `wait_until_<Condition>`
- 對於 Amazon S3，此特徵為 [aws\\_sdk\\_s3::client::Waiters](#)。



下列範例使用 Amazon S3。不過，對於任何已定義一或多個服務生 AWS 服務的，這些概念都是相同的。

下列程式碼範例顯示使用等待程式函數，而不是寫入輪詢邏輯，以等待建立後儲存貯體存在。

```
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

每個等待方法都會傳回一個 `Result<FinalPoll<...>, WaiterError<...>>`，可用來在達到所需條件或錯誤的最終回應時取得。如需詳細資訊，請參閱 Rust API 文件中的 [FinalPoll](#) 和 [WaiterError](#)。

# 適用於 Rust 的 SDK 程式碼範例

本主題中的程式碼範例說明如何搭配 Rust 使用 AWS SDK AWS。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

有些服務包含其他範例類別，示範如何利用服務特定的程式庫或函數。

## 服務

- [使用 SDK for Rust 的 API Gateway 範例](#)
- [使用 SDK for Rust 的 API Gateway Management API 範例](#)
- [使用 SDK for Rust 的 Application Auto Scaling 範例](#)
- [使用 SDK for Rust 的 Aurora 範例](#)
- [使用 SDK for Rust 的 Auto Scaling 範例](#)
- [使用 SDK for Rust 的 Amazon Bedrock 執行期範例](#)
- [使用 SDK for Rust 的 Amazon Cognito Identity Provider 範例](#)
- [使用 SDK for Rust 的 Amazon Cognito Sync 範例](#)
- [使用 SDK for Rust 的 Firehose 範例](#)
- [使用 SDK for Rust 的 Amazon DocumentDB 範例](#)
- [使用 SDK for Rust 的 DynamoDB 範例](#)
- [使用 SDK for Rust 的 Amazon EBS 範例](#)
- [使用 SDK for Rust 的 Amazon EC2 範例](#)
- [使用 SDK for Rust 的 Amazon ECR 範例](#)
- [使用 SDK for Rust 的 Amazon ECS 範例](#)
- [使用 SDK for Rust 的 Amazon EKS 範例](#)
- [AWS Glue 使用 SDK for Rust 的範例](#)
- [使用 SDK for Rust 的 IAM 範例](#)
- [AWS IoT 使用 SDK for Rust 的範例](#)

- [使用 SDK for Rust 的 Kinesis 範例](#)
- [AWS KMS 使用 SDK for Rust 的範例](#)
- [使用 SDK for Rust 的 Lambda 範例](#)
- [使用 SDK for Rust 的 MediaLive 範例](#)
- [使用 SDK for Rust 的 MediaPackage 範例](#)
- [使用 SDK for Rust 的 Amazon MSK 範例](#)
- [使用 SDK for Rust 的 Amazon Polly 範例](#)
- [使用 SDK for Rust 的 QLDB 範例](#)
- [使用 SDK for Rust 的 Amazon RDS 範例](#)
- [使用 SDK for Rust 的 Amazon RDS Data Service 範例](#)
- [使用 SDK for Rust 的 Amazon Rekognition 範例](#)
- [使用 SDK for Rust 的 Route 53 範例](#)
- [使用 SDK for Rust 的 Amazon S3 範例](#)
- [使用 SDK for Rust 的 SageMaker AI 範例](#)
- [使用 SDK for Rust 的 Secrets Manager 範例](#)
- [使用 SDK for Rust 的 Amazon SES API v2 範例](#)
- [使用 SDK for Rust 的 Amazon SNS 範例](#)
- [使用 SDK for Rust 的 Amazon SQS 範例](#)
- [AWS STS 使用 SDK for Rust 的範例](#)
- [使用 SDK for Rust 的 Systems Manager 範例](#)
- [使用 SDK for Rust 的 Amazon Transcribe 範例](#)

## 使用 SDK for Rust 的 API Gateway 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 API Gateway 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

AWS 社群貢獻是由多個團隊所建立和維護的範例 AWS。若要提供意見回饋，請使用連結儲存庫中提供的機制。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

## 主題

- [動作](#)
- [案例](#)
- [AWS 社群貢獻](#)

## 動作

### GetRestApis

下列程式碼範例示範如何使用 GetRestApis。

#### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

顯示 區域中的 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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [GetRestApis](#)。

## 案例

### 建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

#### SDK for Rust

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

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

## AWS 社群貢獻

### 建置和測試無伺服器應用程式

下列程式碼範例示範如何使用 API Gateway 搭配 Lambda 和 DynamoDB 建置和測試無伺服器應用程式

#### SDK for Rust

示範如何使用 Rust SDK 建置和測試由 API Gateway 搭配 Lambda 和 DynamoDB 組成的無伺服器應用程式。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

## 使用 SDK for Rust 的 API Gateway Management API 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 API Gateway Management API 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

### 動作

#### PostToConnection

下列程式碼範例示範如何使用 PostToConnection。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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);
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [PostToConnection](#)。

## 使用 SDK for Rust 的 Application Auto Scaling 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Application Auto Scaling 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

### 主題

- [動作](#)

## 動作

### DescribeScalingPolicies

下列程式碼範例示範如何使用 DescribeScalingPolicies。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeScalingPolicies](#)。

## 使用 SDK for Rust 的 Aurora 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Aurora 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。



每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

## 開始使用

### Hello Aurora

下列程式碼範例示範如何開始使用 Aurora。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeDBClusters](#)。

## 主題

- [基本概念](#)
- [動作](#)

## 基本概念

### 了解基本概念

以下程式碼範例顯示做法：

- 建立自訂 Aurora 資料庫叢集參數群組並設定參數值。
- 建立使用該參數群組的資料庫叢集。
- 建立包含該資料庫的資料庫執行個體。
- 拍攝該資料庫叢集的快照，並清理資源。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

一種程式庫，其中包含 Aurora 案例的案例特定功能。

```
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_engines
        .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;
```

```

        .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;
```

在 RDS 用戶端包裝函式使用自動模擬對程式庫進行測試。

```
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");
}

```

一種二進位檔案，其會使用查詢器從前端執行案例，以便使用者可以做出某些決策。

```

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::
```

```
// 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
}
}
}

```

Amazon RDS 服務的包裝函式，其允許自動模擬以進行測試。

```

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
    }
}
```

在此案例中使用具有相依項的 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"] }
```

- 如需 API 詳細資訊，請參閱 [AWS SDK for Rust API reference](#) 中的下列主題。
  - [CreateDBCluster](#)
  - [CreateDBClusterParameterGroup](#)
  - [CreateDBClusterSnapshot](#)
  - [CreateDBInstance](#)
  - [DeleteDBCluster](#)
  - [DeleteDBClusterParameterGroup](#)
  - [DeleteDBInstance](#)
  - [DescribeDBClusterParameterGroups](#)
  - [DescribeDBClusterParameters](#)
  - [DescribeDBClusterSnapshots](#)
  - [DescribeDBClusters](#)
  - [DescribeDBEngineVersions](#)
  - [DescribeDBInstances](#)
  - [DescribeOrderableDBInstanceOptions](#)
  - [ModifyDBClusterParameterGroup](#)

## 動作

### CreateDBCluster

下列程式碼範例示範如何使用 CreateDBCluster。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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;
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateDBCluster](#)。

## CreateDBClusterParameterGroup

下列程式碼範例示範如何使用 CreateDBClusterParameterGroup。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

// 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());
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateDBClusterParameterGroup](#)。

## CreateDBClusterSnapshot

下列程式碼範例示範如何使用 CreateDBClusterSnapshot。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateDBClusterSnapshot](#)。

## CreateDBInstance

下列程式碼範例示範如何使用 CreateDBInstance。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateDBInstance](#)。

## DeleteDBCluster

下列程式碼範例示範如何使用 DeleteDBCluster。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteDBCluster](#)。

## DeleteDBClusterParameterGroup

下列程式碼範例示範如何使用 DeleteDBClusterParameterGroup。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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())
    })
    .returning(|| Ok(DescribeDbClustersOutput::builder().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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteDBClusterParameterGroup](#)。

## DeleteDBInstance

下列程式碼範例示範如何使用 DeleteDBInstance。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteDBInstance](#)。

## DescribeDBClusterParameters

下列程式碼範例示範如何使用 DescribeDBClusterParameters。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

// 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");
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeDBClusterParameters](#)。

## DescribeDBClusters

下列程式碼範例示範如何使用 DescribeDBClusters。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

// 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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeDBClusters](#)。

## DescribeDBEngineVersions

下列程式碼範例示範如何使用 DescribeDBEngineVersions。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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::

```

```
        )
        .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"
    );
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeDBEngineVersions](#)。

## DescribeDBInstances

下列程式碼範例示範如何使用 DescribeDBInstances。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeDBInstances](#)。

## DescribeOrderableDBInstanceOptions

下列程式碼範例示範如何使用 DescribeOrderableDBInstanceOptions。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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"
);
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeOrderableDBInstanceOptions](#)。

## ModifyDBClusterParameterGroup

下列程式碼範例示範如何使用 ModifyDBClusterParameterGroup。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ModifyDBClusterParameterGroup](#)。

## 使用 SDK for Rust 的 Auto Scaling 範例

下列程式碼範例示範如何使用 AWS SDK for Rust with Auto Scaling 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

開始使用

## Hello Auto Scaling

下列程式碼範例示範如何使用 Auto Scaling。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DescribeAutoScalingGroups](#)。

## 主題

- [基本概念](#)
- [動作](#)

## 基本概念

了解基本概念

以下程式碼範例顯示做法：

- 使用啟動範本和可用區域建立 Amazon EC2 Auto Scaling 群組，並取得執行中執行個體的相關資訊。
- 啟用 Amazon CloudWatch 指標集合。
- 更新群組所需的容量，並等待執行個體啟動。
- 終止 群組中的執行個體。
- 列出因應使用者請求和容量變更而發生的擴展活動。
- 取得 CloudWatch 指標的統計資料，然後清除資源。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
[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\t\t\t\tGroup status:");
        match &self.group {
            Ok(groups) => {
                for status in groups {
                    writeln!(f, "\t\t\t\t\t- {status}")?;
                }
            }
            Err(e) => writeln!(f, "\t\t\t\t\t! - {e}")?,
        }
        writeln!(f, "\t\t\t\t\tInstances:");
        match &self.instances {
            Ok(instances) => {
                for instance in instances {
                    writeln!(f, "\t\t\t\t\t- {instance}")?;
                }
            }
            Err(e) => writeln!(f, "\t\t\t\t\t! {e}")?,
        }

        writeln!(f, "\t\t\t\t\tActivities:");
    }
}
```

```
    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 availability_zones = ec2.describe_availability_zones().send().await;
        if let Err(err) = availability_zones {
            return Err(vec![ScenarioError::new("Failed to find AZs", &err)]);
        }

        let availability_zones: Vec<String> = availability_zones
            .unwrap()
            .availability_zones
            .unwrap_or_default()
            .iter()
            .take(3)
            .map(|z| z.zone_name.clone().unwrap())
            .collect();

        // 1. Create an EC2 launch template that you'll use to create an auto
        scaling group. Bonus: use SDK with EC2.CreateLaunchTemplate to create the launch
        template.
        // * Recommended: InstanceType='t1.micro', ImageId='ami-0ca285d4c2cda3300'
        let create_launch_template = ec2
            .create_launch_template()
            .launch_template_name(LAUNCH_TEMPLATE_NAME)
            .launch_template_data(
                RequestLaunchTemplateData::builder()
                    .instance_type(aws_sdk_ec2::types::InstanceType::T1Micro)
                    .image_id("ami-0ca285d4c2cda3300")
                    .build(),
            )
            .send()
            .await
            .map_err(|err| vec![ScenarioError::new("Failed to create launch
template", &err)]?);

        let launch_template_arn = match create_launch_template.launch_template {
            Some(launch_template) =>
                launch_template.launch_template_id.unwrap_or_default(),
```



```
None => {
    // Try to delete the launch template
    let _ = ec2
        .delete_launch_template()
        .launch_template_name(LAUNCH_TEMPLATE_NAME)
        .send()
        .await;
    return Err(vec![ScenarioError::with("Failed to load launch
template")]);
}
};

// 2. CreateAutoScalingGroup: pass it the launch template you created in
step 0. Give it min/max of 1 instance.
// You can use EC2.describe_availability_zones() to get a list of AZs (you
have to specify an AZ when you create the group).
// Wait for instance to launch. Use a waiter if you have one, otherwise
DescribeAutoScalingInstances until LifecycleState='InService'
if let Err(err) = autoscaling
    .create_auto_scaling_group()
    .auto_scaling_group_name(auto_scaling_group_name.as_str())
    .launch_template(
        LaunchTemplateSpecification::builder()
            .launch_template_id(launch_template_arn.clone())
            .version("$Latest")
            .build(),
    )
    .max_size(1)
    .min_size(1)
    .set_availability_zones(Some(availability_zones))
    .send()
    .await
{
    let mut errs = vec![ScenarioError::new(
        "Failed to create autoscaling group",
        &err,
    )];

    if let Err(err) = autoscaling
        .delete_auto_scaling_group()
        .auto_scaling_group_name(auto_scaling_group_name.as_str())
        .send()
        .await
    {
```

```
        errs.push(ScenarioError::new(
            "Failed to clean up autoscaling group",
            &err,
        ));
    }

    if let Err(err) = ec2
        .delete_launch_template()
        .launch_template_id(launch_template_arn.clone())
        .send()
        .await
    {
        errs.push(ScenarioError::new(
            "Failed to clean up launch template",
            &err,
        ));
    }
    return Err(errs);
}

let scenario = AutoScalingScenario {
    ec2,
    autoscaling: autoscaling.clone(), // Clients are cheap so cloning here
to prevent a move is ok.
    auto_scaling_group_name: auto_scaling_group_name.clone(),
    launch_template_arn,
};

let enable_metrics_collection = autoscaling
    .enable_metrics_collection()
    .auto_scaling_group_name(auto_scaling_group_name.as_str())
    .granularity("1Minute")
    .set_metrics(Some(vec![
        String::from("GroupMinSize"),
        String::from("GroupMaxSize"),
        String::from("GroupDesiredCapacity"),
        String::from("GroupInServiceInstances"),
        String::from("GroupTotalInstances"),
    ]))
    .send()
    .await;

match enable_metrics_collection {
    Ok(_) => Ok(scenario),
```

```
        Err(err) => {
            scenario.clean_scenario().await?;
            Err(vec![ScenarioError::new(
                "Failed to enable metrics collections for group",
                &err,
            )])
        }
    }
}

pub async fn clean_scenario(self) -> Result<(), Vec<ScenarioError>> {
    let _ = self.wait_for_no_scaling().await;
    let delete_group = self
        .autoscaling
        .delete_auto_scaling_group()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .send()
        .await;

    // 14. Delete LaunchTemplate.
    let delete_launch_template = self
        .ec2
        .delete_launch_template()
        .launch_template_id(self.launch_template_arn.clone())
        .send()
        .await;

    let early_exit = match (delete_group, delete_launch_template) {
        (Ok(_), Ok(_)) => Ok(()),
        (Ok(_), Err(e)) => Err(vec![ScenarioError::new(
            "There was an error cleaning the launch template",
            &e,
        )]),
        (Err(e), Ok(_)) => Err(vec![ScenarioError::new(
            "There was an error cleaning the scale group",
            &e,
        )]),
        (Err(e1), Err(e2)) => Err(vec![
            ScenarioError::new("Multiple error cleaning the scenario Scale
Group", &e1),
            ScenarioError::new("Multiple error cleaning the scenario Launch
Template", &e2),
        ]),
    };
};
```

```
if early_exit.is_err() {
    early_exit
} else {
    // Wait for delete_group to finish
    let waiter = Waiter::new();
    let mut errors = Vec::<ScenarioError>::new();
    while errors.len() < 3 {
        if let Err(e) = waiter.sleep().await {
            errors.push(e);
            continue;
        }
        let describe_group = self
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await;
        match describe_group {
            Ok(group) => match group.auto_scaling_groups().first() {
                Some(group) => {
                    if group.status() != Some("Delete in progress") {
                        errors.push(ScenarioError::with(format!(
                            "Group in an unknown state while deleting: {}",
                            group.status().unwrap_or("unknown error")
                        )));
                        return Err(errors);
                    }
                }
                None => return Ok(()),
            },
            Err(err) => {
                errors.push(ScenarioError::new("Failed to describe
autoscaling group during cleanup 3 times, last error", &err));
            }
        }
        if errors.len() > 3 {
            return Err(errors);
        }
    }
    Err(vec![ScenarioError::with(
        "Exited cleanup wait loop without returning success or failing after
three rounds",
    )])
}
```

```
    }
}

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

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

    // 10. DescribeScalingActivities: list the scaling activities that have
    occurred for the group so far.
    // Bonus: use CloudWatch API to get and show some metrics collected for
    the group.
    // CW.ListMetrics with Namespace='AWS/AutoScaling' and
    Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': }]
    // CW.GetMetricStatistics with Statistics='Sum'. Start and End times must
    be in UTC!
    let activities = self
        .autoscaling
        .describe_scaling_activities()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
```

```
        .into_paginator()
        .items()
        .send()
        .collect::
```

```
        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)
}
```

- 如需 API 詳細資訊，請參閱 [AWS SDK for Rust API reference](#) 中的下列主題。
  - [CreateAutoScalingGroup](#)
  - [DeleteAutoScalingGroup](#)

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

## 動作

### CreateAutoScalingGroup

下列程式碼範例示範如何使用 CreateAutoScalingGroup。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateAutoScalingGroup](#)。

## DeleteAutoScalingGroup

下列程式碼範例示範如何使用 DeleteAutoScalingGroup。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteAutoScalingGroup](#)。

## DescribeAutoScalingGroups

下列程式碼範例示範如何使用 DescribeAutoScalingGroups。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DescribeAutoScalingGroups](#)。

## DescribeAutoScalingInstances

下列程式碼範例示範如何使用 DescribeAutoScalingInstances。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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))
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeAutoScalingInstances](#)。

## DescribeScalingActivities

下列程式碼範例示範如何使用 DescribeScalingActivities。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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::
```



```
// 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,
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeScalingActivities](#)。

## DisableMetricsCollection

下列程式碼範例示範如何使用 `DisableMetricsCollection`。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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;
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DisableMetricsCollection](#)。

## EnableMetricsCollection

下列程式碼範例示範如何使用 EnableMetricsCollection。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [EnableMetricsCollection](#)。

## SetDesiredCapacity

下列程式碼範例示範如何使用 SetDesiredCapacity。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [SetDesiredCapacity](#)。

## TerminateInstanceInAutoScalingGroup

下列程式碼範例示範如何使用 TerminateInstanceInAutoScalingGroup。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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"))
    }
}

pub async fn get_group(&self) -> Result<AutoScalingGroup, ScenarioError> {
    let describe_auto_scaling_groups = self
        .autoscaling
```

```
        .describe_auto_scaling_groups()
        .auto_scaling_group_names(self.auto_scaling_group_name.clone())
        .send()
        .await;

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

    let describe_auto_scaling_groups_output =
describe_auto_scaling_groups.unwrap();
    let auto_scaling_groups =
describe_auto_scaling_groups_output.auto_scaling_groups();
    let auto_scaling_group = auto_scaling_groups.first();

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

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [TerminateInstanceInAutoScalingGroup](#)。

## UpdateAutoScalingGroup

下列程式碼範例示範如何使用 UpdateAutoScalingGroup。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [UpdateAutoScalingGroup](#)。

## 使用 SDK for Rust 的 Amazon Bedrock 執行期範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon Bedrock Runtime 來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

### 主題

- [Anthropic Claude](#)

# Anthropic Claude

## Converse

下列程式碼範例示範如何使用 Bedrock 的 Converse API，將文字訊息傳送至 Anthropic Claude。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

使用 Bedrock 的 Converse API，將文字訊息傳送至 Anthropic Claude。

```
#[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)
}

```

使用陳述式、錯誤公用程式和常數。

```

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::ModelErrorTimeoutException(_) => "Model took too long",
            ConverseError::ModelErrorNotReadyException(_) => "Model is not ready",
            _ => "Unknown",
        })
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 AWS Rust 的 SDK API 參考中的 [Converse](#)。

## ConverseStream

下列程式碼範例示範如何使用 Bedrock 的 Converse API 將文字訊息傳送至 Anthropic Claude，並即時處理回應串流。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

使用 Bedrock 的 ConverseStream API，將文字訊息傳送至 Anthropic Claude 並串流回覆字符。

```
#[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(),
    })
}

```

使用陳述式、錯誤公用程式和常數。

```

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(),
    ),
}
}
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ConverseStream](#)。

### 案例：工具搭配 Converse API 使用

下列程式碼範例示範如何在應用程式、生成式 AI 模型和連線工具或 APIs 之間建立典型的互動，以介導 AI 與外部世界之間的互動。它使用將外部天氣 API 連接到 AI 模型的範例，以便根據使用者輸入提供即時天氣資訊。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

示範的主要案例和邏輯。這會協調使用者、Amazon Bedrock Converse API 和天氣工具之間的對話。

```

#[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();
}

```

示範使用的天氣工具。此指令碼定義工具規格，並實作邏輯，以使用 Open-Meteo API 擷取天氣資料。

```

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()?)
    }

```

列印訊息內容區塊的公用程式。

```

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:?})"
        )))
    }
}

```

使用陳述式、錯誤公用程式和常數。

```

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(),
        })
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 AWS Rust 的 SDK API 參考中的 [Converse](#)。

## 使用 SDK for Rust 的 Amazon Cognito Identity Provider 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon Cognito Identity Provider 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [動作](#)

### 動作

#### ListUserPools

下列程式碼範例示範如何使用 ListUserPools。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [ListUserPools](#)。

## 使用 SDK for Rust 的 Amazon Cognito Sync 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon Cognito Sync 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

### 主題

- [動作](#)

## 動作

### ListIdentityPoolUsage

下列程式碼範例示範如何使用 ListIdentityPoolUsage。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [ListIdentityPoolUsage](#)。

## 使用 SDK for Rust 的 Firehose 範例

下列程式碼範例示範如何使用 AWS SDK for Rust with Firehose 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

## 動作

### PutRecordBatch

下列程式碼範例示範如何使用 PutRecordBatch。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [PutRecordBatch](#)。

## 使用 SDK for Rust 的 Amazon DocumentDB 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon DocumentDB 來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [無伺服器範例](#)

### 無伺服器範例

使用 Amazon DocumentDB 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收由從 DocumentDB 變更串流接收記錄所觸發的事件。函數會擷取 DocumentDB 承載並記下記錄內容。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 使用 Amazon DocumentDB 事件。

```
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 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 DynamoDB 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

AWS 社群貢獻是由多個團隊所建立和維護的範例 AWS。若要提供意見回饋，請使用連結儲存庫中提供的機制。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

### 主題

- [動作](#)
- [案例](#)
- [無伺服器範例](#)
- [AWS 社群貢獻](#)

# 動作

## CreateTable

下列程式碼範例示範如何使用 CreateTable。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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 pt = ProvisionedThroughput::builder()
        .read_capacity_units(10)
        .write_capacity_units(5)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
```

```
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .provisioned_throughput(pt)
        .send()
        .await;

match create_table_response {
    Ok(out) => {
        println!("Added table {} with key {}", table, key);
        Ok(out)
    }
    Err(e) => {
        eprintln!("Got an error creating table:");
        eprintln!("{}", e);
        Err(Error::unhandled(e))
    }
}
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateTable](#)。

## DeleteItem

下列程式碼範例示範如何使用 DeleteItem。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)),
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteItem](#)。

## DeleteTable

下列程式碼範例示範如何使用 DeleteTable。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 的詳細資訊，請參閱 [《適用於 Rust 的 AWS SDK API 參考》](#) 中的 DeleteTable。

## ListTables

下列程式碼範例示範如何使用 ListTables。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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::
```

判斷資料表是否存在。

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



```
}  
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListTables](#)。

## PutItem

下列程式碼範例示範如何使用 PutItem。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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,
})
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [PutItem](#)。

## Query

下列程式碼範例示範如何使用 Query。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

尋找在指定年份製作的電影。

```
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![])
    }
}

```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [Query](#)。

## Scan

下列程式碼範例示範如何使用 Scan。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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(())
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [Scan](#)。

## 案例

### 連線至本機執行個體

下列程式碼範例示範如何覆寫端點 URL，以連線至 DynamoDB 和 AWS SDK 的本機開發部署。

如需詳細資訊，請參閱 [DynamoDB Local](#)。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

## 建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

### SDK for Rust

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

### 此範例中使用的服務

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

## 使用 PartiQL 查詢資料表

以下程式碼範例顯示做法：

- 透過執行 SELECT 陳述式取得項目。
- 透過執行 INSERT 陳述式新增項目。
- 透過執行 UPDATE 陳述式更新項目。
- 透過執行 DELETE 陳述式刪除項目。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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");

    let pt = ProvisionedThroughput::builder()
        .read_capacity_units(10)
        .write_capacity_units(5)
        .build()
        .expect("creating ProvisionedThroughput");
```

```

    match client
      .create_table()
      .table_name(table)
      .key_schema(ks)
      .attribute_definitions(ad)
      .provisioned_throughput(pt)
      .send()
      .await
    {
      Ok(_) => Ok(()),
      Err(e) => Err(e),
    }
  }

  async fn add_item(client: &Client, item: Item) -> Result<(),
  SdkError<ExecuteStatementError>> {
    match client
      .execute_statement()
      .statement(format!(
        r#"INSERT INTO "{}" VALUE {{
          "{}": ?,
          "account_type": ?,
          "age": ?,
          "first_name": ?,
          "last_name": ?
        }} "#,
        item.table, item.key
      ))
      .set_parameters(Some(vec![
        AttributeValue::S(item.utype),
        AttributeValue::S(item.age),
        AttributeValue::S(item.first_name),
        AttributeValue::S(item.last_name),
      ]))
      .send()
      .await
    {
      Ok(_) => Ok(()),
      Err(e) => Err(e),
    }
  }

  async fn query_item(client: &Client, item: Item) -> bool {
    match client

```

```

        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {
            println!("Got an error querying table:");
            println!("{}", e);
            process::exit(1);
        }
    }
}

async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{}" WHERE "{}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;
}

```



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

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ExecuteStatement](#)。

## 儲存 EXIF 和其他映像資訊

以下程式碼範例顯示做法：

- 從 JPG、JPEG 或 PNG 檔案中取得 EXIF 資訊。
- 將映像檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Rekognition 識別檔案中的三個主要屬性 (標籤)。
- 將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表。

## SDK for Rust

從 JPG、JPEG 或 PNG 檔案獲取 EXIF 資訊，將映像檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Rekognition 識別三個主要屬性 (Amazon Rekognition 中的標籤)，然後將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表中。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3

## 無伺服器範例

使用 DynamoDB 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收由接收來自 DynamoDB 串流的記錄所觸發的事件。函數會擷取 DynamoDB 承載並記下記錄內容。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 DynamoDB 事件。

```
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(())
}
```

## 使用 DynamoDB 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何為從 DynamoDB 串流接收事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
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 社群貢獻

### 建置和測試無伺服器應用程式

下列程式碼範例示範如何使用 API Gateway 搭配 Lambda 和 DynamoDB 建置和測試無伺服器應用程式

### SDK for Rust

示範如何使用 Rust 開發套件建置和測試由 API Gateway 搭配 Lambda 和 DynamoDB 組成的無伺服器應用程式。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

## 使用 SDK for Rust 的 Amazon EBS 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon EBS 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

## 動作

### CompleteSnapshot

下列程式碼範例示範如何使用 CompleteSnapshot。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [CompleteSnapshot](#)。

## PutSnapshotBlock

下列程式碼範例示範如何使用 PutSnapshotBlock。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [PutSnapshotBlock](#)。

## StartSnapshot

下列程式碼範例示範如何使用 StartSnapshot。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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())  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [StartSnapshot](#)。

## 使用 SDK for Rust 的 Amazon EC2 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon EC2 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。



Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

## 開始使用

### 您好 Amazon EC2

下列程式碼範例示範如何開始使用 Amazon EC2。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DescribeSecurityGroups](#)。

## 主題

- [基本概念](#)
- [動作](#)

## 基本概念

### 了解基本概念

以下程式碼範例顯示做法：

- 建立金鑰對和安全群組。
- 選取 Amazon Machine Image (AMI) 和相容的執行個體類型，然後建立執行個體。
- 停止並重新啟動執行個體。
- 將彈性 IP 地址與您的執行個體建立關聯。
- 使用 SSH 連線至執行個體，然後清理資源。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

EC2InstanceScenario 實作包含邏輯，以執行整體範例。

```
//! 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 結構做為測試的自動模擬點，其函數會包裝 EC2 SDK 呼叫。

```

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 結構做為測試的自動模擬點，其函數會包裝 SSM SDK 呼叫。

```

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)
}
}

```

該案例使用數個「管理員」樣式的結構來處理存取在整個案例中建立和刪除的資源。

```

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."),
        }
    }
}
}

```

案例的主要進入點。

```

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;
}
```

• 如需 API 詳細資訊，請參閱 [AWS SDK for Rust API reference](#) 中的下列主題。

- [AllocateAddress](#)
- [AssociateAddress](#)
- [AuthorizeSecurityGroupIngress](#)
- [CreateKeyPair](#)
- [CreateSecurityGroup](#)
- [DeleteKeyPair](#)
- [DeleteSecurityGroup](#)
- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

## 動作

### AllocateAddress

下列程式碼範例示範如何使用 AllocateAddress。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [AllocateAddress](#)。

### AssociateAddress

下列程式碼範例示範如何使用 AssociateAddress。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [AssociateAddress](#)。

## AuthorizeSecurityGroupIngress

下列程式碼範例示範如何使用 AuthorizeSecurityGroupIngress。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/// 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(())
}

```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [AuthorizeSecurityGroupIngress](#)。

## CreateKeyPair

下列程式碼範例示範如何使用 CreateKeyPair。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

Rust 實作會呼叫 EC2 用戶端的 create\_key\_pair，並擷取傳回的資料。

```

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))
}

```

呼叫 `create_key impl` 並安全儲存 PEM 私有金鑰的函數。

```

/// 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)
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [CreateKeyPair](#)。

## CreateSecurityGroup

下列程式碼範例示範如何使用 CreateSecurityGroup。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
```

```
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [CreateSecurityGroup](#)。

## CreateTags

下列程式碼範例示範如何使用 CreateTags。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

此範例會在建立執行個體後套用名稱標籤。

```
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())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [CreateTags](#)。

## DeleteKeyPair

下列程式碼範例示範如何使用 DeleteKeyPair。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

包裝 `delete_key` 也移除後端私有 PEM 金鑰。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DeleteKeyPair](#)。

## DeleteSecurityGroup

下列程式碼範例示範如何使用 `DeleteSecurityGroup`。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DeleteSecurityGroup](#)。

## DeleteSnapshot

下列程式碼範例示範如何使用 DeleteSnapshot。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

    println!("Deleted");

    Ok(())
}
```

```
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteSnapshot](#)。

## DescribeImages

下列程式碼範例示範如何使用 DescribeImages。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
    }
}
```

使用 list\_images 函數搭配 SSM，根據您的環境進行限制。如需 SSM 的詳細資訊，請參閱 [https://docs.aws.amazon.com/systems-manager/latest/userguide/example\\_ssm\\_GetParameters\\_section.html](https://docs.aws.amazon.com/systems-manager/latest/userguide/example_ssm_GetParameters_section.html)。

```
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)
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeImages](#)。

## DescribeInstanceStatus

下列程式碼範例示範如何使用 DescribeInstanceStatus。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeInstanceStatus](#)。

## DescribeInstanceTypes

下列程式碼範例示範如何使用 DescribeInstanceTypes。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DescribeInstanceTypes](#)。

## DescribeInstances

下列程式碼範例示範如何使用 DescribeInstances。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

擷取 EC2 執行個體的詳細資訊。

```
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())
}
```

建立 EC2 執行個體之後，請擷取並存放其詳細資訊。

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

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeInstances](#)。

## DescribeKeyPairs

下列程式碼範例示範如何使用 DescribeKeyPairs。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeKeyPairs](#)。

## DescribeRegions

下列程式碼範例示範如何使用 DescribeRegions。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeRegions](#)。

## DescribeSecurityGroups

下列程式碼範例示範如何使用 DescribeSecurityGroups。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)", code, message);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DescribeSecurityGroups](#)。

## DescribeSnapshots

下列程式碼範例示範如何使用 DescribeSnapshots。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

顯示快照的狀態。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeSnapshots](#)。

## DisassociateAddress

下列程式碼範例示範如何使用 DisassociateAddress。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DisassociateAddress](#)。

## RebootInstances

下列程式碼範例示範如何使用 RebootInstances。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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 Waiters API，讓執行個體處於已停止和就緒狀態的等待者。使用 Waiters API 需要在 rust 檔案中使用 `use aws\_sdk\_ec2 : : client : : Waiters`。

```

/// 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(())
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [RebootInstances](#)。

## ReleaseAddress

下列程式碼範例示範如何使用 ReleaseAddress。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

pub async fn deallocate_ip_address(&self, allocation_id: &str) -> Result<(),
EC2Error> {

```

```
        self.client
            .release_address()
            .allocation_id(allocation_id)
            .send()
            .await?;
        Ok(())
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ReleaseAddress](#)。

## RunInstances

下列程式碼範例示範如何使用 RunInstances。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [RunInstances](#)。

## StartInstances

下列程式碼範例示範如何使用 StartInstances。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

依執行個體 ID 啟動 EC2 執行個體。

```
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 Waiters API，等待執行個體處於就緒狀態和狀態良好狀態。使用 Waiters API 需要在 rust 檔案中使用 `use aws\_sdk\_ec2 : : client : : Waiters`。

```
/// 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(())
}

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [StartInstances](#)。

## StopInstances

下列程式碼範例示範如何使用 StopInstances。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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 Waiters API 等待執行個體處於停止狀態。使用 Waiters API 需要在 rust 檔案中使用 `use aws\_sdk\_ec2 : : client : : Waiters`。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [StopInstances](#)。

## TerminateInstances

下列程式碼範例示範如何使用 TerminateInstances。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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 Waiters API，等待執行個體處於終止狀態。使用 Waiters API 需要在 rust 檔案中使用 `use aws\_sdk\_ec2::client::Waiters`。

```

    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(())
    }

```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [TerminateInstances](#)。

## 使用 SDK for Rust 的 Amazon ECR 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon ECR 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

## 主題

- [動作](#)

## 動作

### DescribeRepositories

下列程式碼範例示範如何使用 DescribeRepositories。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeRepositories](#)。

### ListImages

下列程式碼範例示範如何使用 ListImages。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListImages](#)。

## 使用 SDK for Rust 的 Amazon ECS 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon ECS 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [動作](#)

## 動作

### CreateCluster

下列程式碼範例示範如何使用 CreateCluster。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [CreateCluster](#)。

### DeleteCluster

下列程式碼範例示範如何使用 DeleteCluster。



## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DeleteCluster](#)。

## DescribeClusters

下列程式碼範例示範如何使用 DescribeClusters。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeClusters](#)。

## 使用 SDK for Rust 的 Amazon EKS 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon EKS 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [動作](#)

### 動作

#### CreateCluster

下列程式碼範例示範如何使用 CreateCluster。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [CreateCluster](#)。

## DeleteCluster

下列程式碼範例示範如何使用 DeleteCluster。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DeleteCluster](#)。

## AWS Glue 使用 SDK for Rust 的範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 來執行動作和實作常見案例 AWS Glue。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

開始使用

您好 AWS Glue

下列程式碼範例示範如何開始使用 AWS Glue。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)),
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [ListJobs](#)。

## 主題

- [基本概念](#)
- [動作](#)

## 基本概念

### 了解基本概念

以下程式碼範例顯示做法：

- 建立網路爬取公有 Amazon S3 儲存貯體的爬蟲程式，以及產生 CSV 格式中繼資料的資料庫。
- 列出 中資料庫和資料表的相關資訊 AWS Glue Data Catalog。
- 建立從 S3 儲存貯體中擷取 CSV 資料的任務、轉換資料，以及將 JSON 格式的輸出載入至另一個 S3 儲存貯體。
- 列出任務執行的相關資訊、檢視已轉換的資料以及清除資源。

如需詳細資訊，請參閱[教學課程：AWS Glue Studio 入門](#)。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立並執行可網路爬取公有 Amazon Simple Storage Service (Amazon S3) 儲存貯體的爬蟲程式，並產生描述其所尋找 CSV 格式資料的中繼資料的資料庫。

```
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 中資料庫和資料表的相關資訊 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();
```

建立並執行從來源 Amazon S3 儲存貯體中擷取 CSV 資料的任務、透過移除和重新命名欄位進行轉換，以及將 JSON 格式的輸出載入另一個 Amazon S3 儲存貯體。

```
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();
```

刪除透過示範建立的所有資源。

```
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)?;
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的下列主題。
  - [CreateCrawler](#)
  - [CreateJob](#)
  - [DeleteCrawler](#)
  - [DeleteDatabase](#)
  - [DeleteJob](#)
  - [DeleteTable](#)
  - [GetCrawler](#)
  - [GetDatabase](#)
  - [GetDatabases](#)
  - [GetJob](#)
  - [GetJobRun](#)

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

## 動作

### CreateCrawler

下列程式碼範例示範如何使用 CreateCrawler。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [CreateCrawler](#)。

## CreateJob

下列程式碼範例示範如何使用 CreateJob。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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())
})??;

```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [CreateJob](#)。

## DeleteCrawler

下列程式碼範例示範如何使用 DeleteCrawler。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [DeleteCrawler](#)。

## DeleteDatabase

下列程式碼範例示範如何使用 DeleteDatabase。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
glue.delete_database()
```

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

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [DeleteDatabase](#)。

## DeleteJob

下列程式碼範例示範如何使用 DeleteJob。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [DeleteJob](#)。

## DeleteTable

下列程式碼範例示範如何使用 DeleteTable。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 的詳細資訊，請參閱 [《適用於 Rust 的 AWS SDK API 參考》](#) 中的 DeleteTable。

## GetCrawler

下列程式碼範例示範如何使用 GetCrawler。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
let tmp_crawler = glue
    .get_crawler()
    .name(self.crawler())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- 如需 API 詳細資訊，請參閱 [《適用於 Rust 的 AWS SDK API 參考》](#) 中的 [GetCrawler](#)。

## GetDatabase

下列程式碼範例示範如何使用 GetDatabase。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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()))?;
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [GetDatabase](#)。

## GetJobRun

下列程式碼範例示範如何使用 GetJobRun。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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();
}

```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [GetJobRun](#)。

## GetTables

下列程式碼範例示範如何使用 GetTables。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
let tables = glue
```



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

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

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [GetTables](#)。

## ListJobs

下列程式碼範例示範如何使用 ListJobs。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)),
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [ListJobs](#)。

## StartCrawler

下列程式碼範例示範如何使用 StartCrawler。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [StartCrawler](#)。

## StartJobRun

下列程式碼範例示範如何使用 StartJobRun。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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();
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [StartJobRun](#)。

## 使用 SDK for Rust 的 IAM 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 IAM 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。


每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

開始使用

Hello IAM

下列程式碼範例示範如何開始使用 IAM。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

來自 `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(())
}
```

來自 `src/iam-service-lib.rs`。

```
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)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListPolicies](#)。

## 主題

- [基本概念](#)
- [動作](#)

## 基本概念

### 了解基本概念

下列程式碼範例示範如何建立使用者並擔任角色。

**⚠ Warning**

為避免安全風險，在開發專用軟體或使用真實資料時，請勿使用 IAM 使用者進行身分驗證。相反地，搭配使用聯合功能和身分提供者，例如 [AWS IAM Identity Center](#)。

- 建立沒有許可的使用者。
- 建立一個可授予許可的角色，以列出帳戶的 Amazon S3 儲存貯體。
- 新增政策，讓使用者擔任該角色。
- 使用暫時憑證，擔任角色並列出 Amazon S3 儲存貯體，然後清理資源。

## SDK for Rust

**i Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的下列主題。
  - [AttachRolePolicy](#)
  - [CreateAccessKey](#)
  - [CreatePolicy](#)
  - [CreateRole](#)
  - [CreateUser](#)
  - [DeleteAccessKey](#)
  - [DeletePolicy](#)
  - [DeleteRole](#)
  - [DeleteUser](#)
  - [DeleteUserPolicy](#)
  - [DetachRolePolicy](#)
  - [PutUserPolicy](#)

## 動作

### AttachRolePolicy

下列程式碼範例示範如何使用 `AttachRolePolicy`。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [AttachRolePolicy](#)。

### AttachUserPolicy

下列程式碼範例示範如何使用 `AttachUserPolicy`。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [AttachUserPolicy](#)。

## CreateAccessKey

下列程式碼範例示範如何使用 CreateAccessKey。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [CreateAccessKey](#)。

## CreatePolicy

下列程式碼範例示範如何使用 CreatePolicy。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [CreatePolicy](#)。

## CreateRole

下列程式碼範例示範如何使用 CreateRole。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [CreateRole](#)。

## CreateServiceLinkedRole

下列程式碼範例示範如何使用 CreateServiceLinkedRole。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [CreateServiceLinkedRole](#)。

## CreateUser

下列程式碼範例示範如何使用 CreateUser。



## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [CreateUser](#)。

## DeleteAccessKey

下列程式碼範例示範如何使用 DeleteAccessKey。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [DeleteAccessKey](#)。

## DeletePolicy

下列程式碼範例示範如何使用 DeletePolicy。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
pub async fn delete_policy(client: &iamClient, policy: Policy) -> Result<(),
iamError> {
    client
        .delete_policy()
        .policy_arn(policy.arn.unwrap())
        .send()
        .await?;
    Ok(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [DeletePolicy](#)。

## DeleteRole

下列程式碼範例示範如何使用 DeleteRole。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [DeleteRole](#)。

## DeleteServiceLinkedRole

下列程式碼範例示範如何使用 DeleteServiceLinkedRole。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [DeleteServiceLinkedRole](#)。

## DeleteUser

下列程式碼範例示範如何使用 DeleteUser。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [DeleteUser](#)。

## DeleteUserPolicy

下列程式碼範例示範如何使用 DeleteUserPolicy。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [DeleteUserPolicy](#)。

## DetachRolePolicy

下列程式碼範例示範如何使用 DetachRolePolicy。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [DetachRolePolicy](#)。

## DetachUserPolicy

下列程式碼範例示範如何使用 DetachUserPolicy。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [DetachUserPolicy](#)。

## GetAccountPasswordPolicy

下列程式碼範例示範如何使用 GetAccountPasswordPolicy。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
pub async fn get_account_password_policy(
    client: &iamClient,
) -> Result<GetAccountPasswordPolicyOutput, SdkError<GetAccountPasswordPolicyError>>
{
    let response = client.get_account_password_policy().send().await?;

    Ok(response)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [GetAccountPasswordPolicy](#)。

## GetRole

下列程式碼範例示範如何使用 GetRole。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```



```
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [GetRole](#)。

## ListAttachedRolePolicies

下列程式碼範例示範如何使用 ListAttachedRolePolicies。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [ListAttachedRolePolicies](#)。

## ListGroups

下列程式碼範例示範如何使用 ListGroups。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [ListGroups](#)。

## ListPolicies

下列程式碼範例示範如何使用 ListPolicies。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [ListPolicies](#)。

## ListRolePolicies

下列程式碼範例示範如何使用 ListRolePolicies。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [ListRolePolicies](#)。

## ListRoles

下列程式碼範例示範如何使用 ListRoles。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [ListRoles](#)。

## ListSAMLProviders

下列程式碼範例示範如何使用 ListSAMLProviders。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
pub async fn list_saml_providers(
```

```
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [ListSAMLProviders](#)。

## ListUsers

下列程式碼範例示範如何使用 ListUsers。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的 [ListUsers](#)。

# AWS IoT 使用 SDK for Rust 的範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 來執行動作和實作常見案例 AWS IoT。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

## 動作

### DescribeEndpoint

下列程式碼範例示範如何使用 DescribeEndpoint。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeEndpoint](#)。

## ListThings

下列程式碼範例示範如何使用 ListThings。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```



- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListThings](#)。

## 使用 SDK for Rust 的 Kinesis 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Kinesis 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

### 主題

- [動作](#)
- [無伺服器範例](#)

## 動作

### CreateStream

下列程式碼範例示範如何使用 CreateStream。

#### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [CreateStream](#)。

## DeleteStream

下列程式碼範例示範如何使用 DeleteStream。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DeleteStream](#)。

## DescribeStream

下列程式碼範例示範如何使用 DescribeStream。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeStream](#)。

## ListStreams

下列程式碼範例示範如何使用 ListStreams。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListStreams](#)。

## PutRecord

下列程式碼範例示範如何使用 PutRecord。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [PutRecord](#)。

## 無伺服器範例

使用 Kinesis 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收由接收來自 Kinesis 串流的記錄所觸發的事件。此函數會擷取 Kinesis 承載、從 Base64 解碼，並記錄記錄內容。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 Kinesis 事件。

```
// 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
}
```

## 使用 Kinesis 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何為從 Kinesis 串流接收事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

## 使用 Rust 搭配 Lambda 來報告 Kinesis 批次項目失敗。

```
// 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 SDK for Rust 的範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 來執行動作和實作常見案例 AWS KMS。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。



每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

## 動作

### CreateKey

下列程式碼範例示範如何使用 CreateKey。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [CreateKey](#)。

### Decrypt

下列程式碼範例示範如何使用 Decrypt。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [解密](#)。

## Encrypt

下列程式碼範例示範如何使用 Encrypt。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [加密](#)。

## GenerateDataKey

下列程式碼範例示範如何使用 GenerateDataKey。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [GenerateDataKey](#)。

## GenerateDataKeyWithoutPlaintext

下列程式碼範例示範如何使用 GenerateDataKeyWithoutPlaintext。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [GenerateDataKeyWithoutPlaintext](#)。

## GenerateRandom

下列程式碼範例示範如何使用 GenerateRandom。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [GenerateRandom](#)。

## ListKeys

下列程式碼範例示範如何使用 ListKeys。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListKeys](#)。

## ReEncrypt

下列程式碼範例示範如何使用 ReEncrypt。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```



- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ReEncrypt](#)。

## 使用 SDK for Rust 的 Lambda 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Lambda 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

AWS 社群貢獻是由多個團隊所建立和維護的範例 AWS。若要提供意見回饋，請使用連結儲存庫中提供的機制。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [基本概念](#)
- [動作](#)
- [案例](#)
- [無伺服器範例](#)
- [AWS 社群貢獻](#)

## 基本概念

了解基本概念

以下程式碼範例顯示做法：

- 建立 IAM 角色和 Lambda 函數，然後上傳處理常式程式碼。
- 調用具有單一參數的函數並取得結果。
- 更新函數程式碼並使用環境變數進行設定。
- 調用具有新參數的函數並取得結果。顯示傳回的執行日誌。
- 列出您帳戶的函數，然後清理相關資源。

如需詳細資訊，請參閱[使用主控台建立 Lambda 函數](#)。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

在此案例中使用具有相依項的 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"
```

一個公用程式集合，可簡化此案例的 Lambda 呼叫。此檔案是套件中的 src/actions.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::Provided2)
        .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(),
    );
}
}
```

從前端到後端執行該案例的二進位檔案，使用命令列旗標來控制某些行為。此檔案是套件中的 `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");
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的下列主題。
- [CreateFunction](#)

- [DeleteFunction](#)
- [GetFunction](#)
- [Invoke](#)
- [ListFunctions](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

## 動作

### CreateFunction

下列程式碼範例示範如何使用 CreateFunction。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/**
 * 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())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [CreateFunction](#)。

## DeleteFunction

下列程式碼範例示範如何使用 DeleteFunction。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/** 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)
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [DeleteFunction](#)。

## GetFunction

下列程式碼範例示範如何使用 GetFunction。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/** 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)
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [GetFunction](#)。

## Invoke

下列程式碼範例示範如何使用 Invoke。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/** 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")
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [Invoke](#)。

## ListFunctions

下列程式碼範例示範如何使用 ListFunctions。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/** 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)
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [ListFunctions](#)。

## UpdateFunctionCode

下列程式碼範例示範如何使用 UpdateFunctionCode。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/** 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())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [UpdateFunctionCode](#)。

## UpdateFunctionConfiguration

下列程式碼範例示範如何使用 UpdateFunctionConfiguration。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/** 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)
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 [UpdateFunctionConfiguration](#)。

## 案例

### 建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

#### SDK for Rust

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

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

## 無伺服器範例

連線至 Lambda 函數中的 Amazon RDS 資料庫

下列程式碼範例示範如何實作連線至 RDS 資料庫的 Lambda 函數。該函數會提出簡單的資料庫請求並傳回結果。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 連線至 Lambda 函數中的 Amazon RDS 資料庫。

```
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 Kinesis 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收由接收來自 Kinesis 串流的記錄所觸發的事件。此函數會擷取 Kinesis 承載、從 Base64 解碼，並記錄記錄內容。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 Kinesis 事件。

```
// 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
}
```

## 使用 DynamoDB 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收由接收來自 DynamoDB 串流的記錄所觸發的事件。函數會擷取 DynamoDB 承載並記下記錄內容。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 DynamoDB 事件。

```
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(())
}
```

## 使用 Amazon DocumentDB 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收由從 DocumentDB 變更串流接收記錄所觸發的事件。函數會擷取 DocumentDB 承載並記下記錄內容。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 使用 Amazon DocumentDB 事件。

```
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 Amazon MSK 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收從 Amazon MSK 叢集接收記錄所觸發的事件。函數會擷取 MSK 承載並記下記錄內容。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 使用 Amazon MSK 事件。

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
```

```
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/aws-labs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error> {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

            let record_text = record.value.as_ref().ok_or("Value is None")?;
            info!("Record: {}", &record_text);

            // perform Base64 decoding
            let record_bytes = BASE64_STANDARD.decode(record_text)?;
            let message = std::str::from_utf8(&record_bytes)?;

            info!("Message: {}", message);
        }

    }

    Ok(()).into()
}

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

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```



## 使用 Amazon S3 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收透過將物件上傳至 S3 儲存貯體所觸發的事件。此函數會從事件參數擷取 S3 儲存貯體名稱和物件金鑰，並呼叫 Amazon S3 API 以擷取和記錄物件的內容類型。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 S3 事件。

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

## 使用 Amazon SNS 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，以接收從 SNS 主題接收訊息所觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 SNS 事件。

```
// 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 record 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 Amazon SQS 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，以接收從 SQS 佇列接收訊息所觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 SQS 事件。

```
// 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
}

```

## 使用 Kinesis 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何為從 Kinesis 串流接收事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

## 使用 Rust 搭配 Lambda 來報告 Kinesis 批次項目失敗。

```

// 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
}
```

## 使用 DynamoDB 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何為從 DynamoDB 串流接收事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

## 使用 Rust 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
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
}
```

## 使用 Amazon SQS 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何為從 SQS 佇列接收事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

## 使用 Rust 搭配 Lambda 報告 SQS 批次項目失敗。

```
// 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 社群貢獻

### 建置和測試無伺服器應用程式

下列程式碼範例示範如何使用 API Gateway 搭配 Lambda 和 DynamoDB 建置和測試無伺服器應用程式

### SDK for Rust

示範如何使用 Rust 開發套件建置和測試由 API Gateway 搭配 Lambda 和 DynamoDB 組成的無伺服器應用程式。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

## 使用 SDK for Rust 的 MediaLive 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 MediaLive 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

## 動作

### ListInputs

下列程式碼範例示範如何使用 ListInputs。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

在 區域中列出您的 MediaLive 輸入名稱和 ARNs。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListInputs](#)。

## 使用 SDK for Rust 的 MediaPackage 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 MediaPackage 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

### 動作

#### ListChannels

下列程式碼範例示範如何使用 ListChannels。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

列出頻道 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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListChannels](#)。

## ListOriginEndpoints

下列程式碼範例示範如何使用 ListOriginEndpoints。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

列出您的端點描述和 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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListOriginEndpoints](#)。

## 使用 SDK for Rust 的 Amazon MSK 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon MSK 來執行動作和實作常見案例。每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [無伺服器範例](#)

### 無伺服器範例

使用 Amazon MSK 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數接收從 Amazon MSK 叢集接收記錄所觸發的事件。函數會擷取 MSK 承載並記下記錄內容。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 使用 Amazon MSK 事件。

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
```

```
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/aws-labs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error> {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

            let record_text = record.value.as_ref().ok_or("Value is None")?;
            info!("Record: {}", &record_text);

            // perform Base64 decoding
            let record_bytes = BASE64_STANDARD.decode(record_text)?;
            let message = std::str::from_utf8(&record_bytes)?;

            info!("Message: {}", message);
        }

    }

    Ok(()).into()
}

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

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");
}
```

```
run(service_fn(function_handler)).await  
}
```

## 使用 SDK for Rust 的 Amazon Polly 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon Polly 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [動作](#)
- [案例](#)

## 動作

### DescribeVoices

下列程式碼範例示範如何使用 DescribeVoices。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [DescribeVoices](#)。

## ListLexicons

下列程式碼範例示範如何使用 ListLexicons。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListLexicons](#)。

## PutLexicon

下列程式碼範例示範如何使用 PutLexicon。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [PutLexicon](#)。

## SynthesizeSpeech

下列程式碼範例示範如何使用 SynthesizeSpeech。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [SynthesizeSpeech](#)。

## 案例

將文字轉換為語音然後返回文字

以下程式碼範例顯示做法：

- 使用 Amazon Polly 將純文字 (UTF-8) 輸入檔案合成至音訊檔案中。
- 將音訊檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Transcribe 將音訊檔案轉換為文字。
- 顯示文字。

### SDK for Rust

使用 Amazon Polly 將純文字 (UTF-8) 輸入檔案合成至音訊檔案中，將音訊檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Transcribe 將該音訊檔案轉換為文字，然後顯示文字。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Polly
- Amazon S3
- Amazon Transcribe

## 使用 SDK for Rust 的 QLDB 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 QLDB 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

## 動作

### CreateLedger

下列程式碼範例示範如何使用 CreateLedger。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
async fn make_ledger(client: &Client, ledger: &str) -> Result<(), Error> {
    let result = client
        .create_ledger()
        .name(ledger)
        .permissions_mode(PermissionsMode::AllowAll)
        .send()
        .await?;
```

```
println!("ARN: {}", result.arn().unwrap());

Ok(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [CreateLedger](#)。

## ListLedgers

下列程式碼範例示範如何使用 ListLedgers。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
async fn show_ledgers(client: &QLDBClient) -> Result<(), Error> {
    let mut pages = client.list_ledgers().into_paginator().page_size(2).send();

    while let Some(page) = pages.next().await {
        println!("* {:?}", page); //Prints an entire page of ledgers.
        for ledger in page.unwrap().ledgers() {
            println!("* {:?}", ledger); //Prints the LedgerSummary of a single
ledger.
        }
    }

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListLedgers](#)。

## 使用 SDK for Rust 的 Amazon RDS 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon RDS 來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

## 主題

- [無伺服器範例](#)

## 無伺服器範例

連線至 Lambda 函數中的 Amazon RDS 資料庫

下列程式碼範例示範如何實作連線至 RDS 資料庫的 Lambda 函數。該函數會提出簡單的資料庫請求並傳回結果。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 連線至 Lambda 函數中的 Amazon RDS 資料庫。

```
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 Data Service 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon RDS Data Service 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

## 動作

### ExecuteStatement

下列程式碼範例示範如何使用 ExecuteStatement。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ExecuteStatement](#)。

## 使用 SDK for Rust 的 Amazon Rekognition 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon Rekognition 來執行動作和實作常見案例。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [案例](#)

### 案例

建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

SDK for Rust

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

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

偵測映像中的人臉

以下程式碼範例顯示做法：

- 在 Amazon S3 儲存貯體儲存映像。
- 使用 Amazon Rekognition 偵測面部細節，例如年齡範圍、性別和情感 (例如微笑)。
- 顯示這些詳細資訊。

適用於 Rust 的 SDK

將映像儲存在 Amazon S3 儲存貯體中，並包含上傳字首，使用 Amazon Rekognition 偵測面部細節，例如年齡範圍、性別和情感 (微笑等)，並顯示這些詳細資訊。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3

儲存 EXIF 和其他映像資訊

以下程式碼範例顯示做法：

- 從 JPG、JPEG 或 PNG 檔案中取得 EXIF 資訊。
- 將映像檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Rekognition 識別檔案中的三個主要屬性 (標籤)。
- 將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表。

## SDK for Rust

從 JPG、JPEG 或 PNG 檔案獲取 EXIF 資訊，將映像檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Rekognition 識別三個主要屬性 (Amazon Rekognition 中的標籤)，然後將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表中。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3

## 使用 SDK for Rust 的 Route 53 範例

下列程式碼範例示範如何使用適用於 Rust 的 AWS SDK 搭配 Route 53 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

## 動作

### ListHostedZones

下列程式碼範例示範如何使用 ListHostedZones。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListHostedZones](#)。

## 使用 SDK for Rust 的 Amazon S3 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon S3 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。


每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

您好 Amazon S3

下列程式碼範例示範如何開始使用 Amazon S3。

SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/// 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(())
}

```

### S3ExampleError 公用程式。

```

/// 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)
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListBuckets](#)。

## 主題

- [基本概念](#)
- [動作](#)
- [案例](#)
- [無伺服器範例](#)

## 基本概念

### 了解基本概念

以下程式碼範例顯示做法：

- 建立儲存貯體並上傳檔案到該儲存貯體。
- 從儲存貯體下載物件。
- 將物件複製至儲存貯體中的子文件夾。
- 列出儲存貯體中的物件。
- 刪除儲存貯體物件和該儲存貯體。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。



```

    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(())
}

```

案例使用的常見動作。

```

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))
            }
        }
    }
}
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的下列主題。
  - [CopyObject](#)
  - [CreateBucket](#)
  - [DeleteBucket](#)
  - [DeleteObjects](#)
  - [GetObject](#)
  - [ListObjectsV2](#)



- [PutObject](#)

## 動作

### CompleteMultipartUpload

下列程式碼範例示範如何使用 CompleteMultipartUpload。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CompleteMultipartUpload](#)。

## CopyObject

下列程式碼範例示範如何使用 CopyObject。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CopyObject](#)。

## CreateBucket

下列程式碼範例示範如何使用 CreateBucket。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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))
        }
    })
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateBucket](#)。

## CreateMultipartUpload

下列程式碼範例示範如何使用 CreateMultipartUpload。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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?;
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateMultipartUpload](#)。

## DeleteBucket

下列程式碼範例示範如何使用 DeleteBucket。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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))
            }
        }
    }
}
```

```
    }  
  }  
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteBucket](#)。

## DeleteObject

下列程式碼範例示範如何使用 DeleteObject。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteObject](#)。



## DeleteObjects

下列程式碼範例示範如何使用 DeleteObjects。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DeleteObjects](#)。

## GetBucketLocation

下列程式碼範例示範如何使用 GetBucketLocation。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [GetBucketLocation](#)。

## GetObject

下列程式碼範例示範如何使用 GetObject。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [GetObject](#)。

## ListBuckets

下列程式碼範例示範如何使用 ListBuckets。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListBuckets](#)。

## ListObjectVersions

下列程式碼範例示範如何使用 ListObjectVersions。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListObjectVersions](#)。

## ListObjectsV2

下列程式碼範例示範如何使用 ListObjectsV2。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListObjectsV2](#)。

## PutObject

下列程式碼範例示範如何使用 PutObject。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [PutObject](#)。

## UploadPart

下列程式碼範例示範如何使用 UploadPart。



## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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?;
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [UploadPart](#)。

## 案例

將文字轉換為語音然後返回文字

以下程式碼範例顯示做法：

- 使用 Amazon Polly 將純文字 (UTF-8) 輸入檔案合成至音訊檔案中。
- 將音訊檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Transcribe 將音訊檔案轉換為文字。
- 顯示文字。

## SDK for Rust

使用 Amazon Polly 將純文字 (UTF-8) 輸入檔案合成至音訊檔案中，將音訊檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Transcribe 將該音訊檔案轉換為文字，然後顯示文字。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Polly
- Amazon S3
- Amazon Transcribe

## 建立預先簽章 URL

下列程式碼範例示範如何建立 Amazon S3 的預先簽章 URL 並上傳物件。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立對 GET S3 物件的預先簽署請求。

```
/// 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(())
}

```

建立對 PUT S3 物件的預先簽章請求。

```

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())
}

```

## 建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

### SDK for Rust

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

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

### 偵測映像中的人臉

以下程式碼範例顯示做法：

- 在 Amazon S3 儲存貯體儲存映像。
- 使用 Amazon Rekognition 偵測面部細節，例如年齡範圍、性別和情感 (例如微笑)。
- 顯示這些詳細資訊。

### 適用於 Rust 的 SDK

將映像儲存在 Amazon S3 儲存貯體中，並包含上傳字首，使用 Amazon Rekognition 偵測面部細節，例如年齡範圍、性別和情感 (微笑等)，並顯示這些詳細資訊。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3

## 從儲存貯體中取得物件 (如果其已修改的話)

下列程式碼範例示範如何從 S3 儲存貯體中物件讀取資料，但僅在自上次擷取時後該儲存貯體尚未修改時才能進行此讀取。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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 tuple containing
// the `last_modified` and `e_tag_1` properties. This is not the expected
// result.
//
// The expected result of the second call to `HeadObject` is an
// `SdkError::ServiceError` containing the HTTP error response. If that's
// the case and the HTTP status is 304 (not modified), the output is a
// tuple containing the values of the HTTP `last-modified` and `etag`
// headers.
//
// If any other HTTP error occurred, the error is returned as an
// `SdkError::ServiceError`.

let (last_modified, e_tag_2) = match head_object_output {
    Ok(head_object) => (
```



```

        Ok(head_object.last_modified().cloned().unwrap()),
        head_object.e_tag.unwrap(),
    ),
    Err(err) => match err {
        SdkError::ServiceError(err) => {
            // Get the raw HTTP response. If its status is 304, the
            // object has not changed. This is the expected code path.
            let http = err.raw();
            match http.status().as_u16() {
                // If the HTTP status is 304: Not Modified, return a
                // tuple containing the values of the HTTP
                // `last-modified` and `etag` headers.
                304 => (
                    Ok(DateTime::from_str(
                        http.headers().get("last-modified").unwrap(),
                        DateTimeFormat::HttpDate,
                    )
                    .unwrap()),
                    http.headers().get("etag").map(|t| t.into()).unwrap(),
                ),
                // Any other HTTP status code is returned as an
                // `SdkError::ServiceError`.
                _ => (Err(SdkError::ServiceError(err)), String::new()),
            }
        }
        // Any other kind of error is returned in a tuple containing the
        // error and an empty string.
        _ => (Err(err), String::new()),
    },
};

warn!("last modified: {last_modified:?}");
assert_eq!(
    e_tag_1, e_tag_2,
    "PutObject and second HeadObject had different eTags"
);

println!("Second value of last modified: {last_modified:?}");
println!("Second tag: {}", e_tag_2);

// Clean up by deleting the object and the bucket.
client
    .delete_object()
    .bucket(bucket_name.as_str())

```

```
        .key(KEY)
        .send()
        .await?;

    client
        .delete_bucket()
        .bucket(bucket_name.as_str())
        .send()
        .await?;

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [GetObject](#)。

## 儲存 EXIF 和其他映像資訊

以下程式碼範例顯示做法：

- 從 JPG、JPEG 或 PNG 檔案中取得 EXIF 資訊。
- 將映像檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Rekognition 識別檔案中的三個主要屬性 (標籤)。
- 將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表。

## SDK for Rust

從 JPG、JPEG 或 PNG 檔案獲取 EXIF 資訊，將映像檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Rekognition 識別三個主要屬性 (Amazon Rekognition 中的標籤)，然後將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表中。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

### 此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3

## 使用 SDK 進行單元和整合測試

下列程式碼範例示範如何使用 AWS SDK 撰寫單元和整合測試時，如何提供最佳實務技術的範例。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

Cargo.toml 用於測試範例。

```
[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"
```

使用自動模擬和服務包裝函式的單元測試範例。

```
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 `StaticReplayClient` 的整合測試範例。

```

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(&[]);
}
}

```

## 上傳或下載大型檔案

下列程式碼範例示範如何上傳或下載大型檔案至 Amazon S3 或從 Amazon S3 下載大型檔案。

如需詳細資訊，請參閱[使用分段上傳以上傳物件](#)。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

## 無伺服器範例

### 使用 Amazon S3 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，以接收透過將物件上傳至 S3 儲存貯體所觸發的事件。此函數會從事件參數擷取 S3 儲存貯體名稱和物件金鑰，並呼叫 Amazon S3 API 以擷取和記錄物件的內容類型。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 S3 事件。

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

## 使用 SDK for Rust 的 SageMaker AI 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 SageMaker AI 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

### 主題

- [動作](#)



## 動作

### ListNotebookInstances

下列程式碼範例示範如何使用 ListNotebookInstances。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [ListNotebookInstances](#)。

### ListTrainingJobs

下列程式碼範例示範如何使用 ListTrainingJobs。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListTrainingJobs](#)。

## 使用 SDK for Rust 的 Secrets Manager 範例

下列程式碼範例示範如何使用 AWS SDK for Rust with Secrets Manager 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)

### 動作

#### GetSecretValue

下列程式碼範例示範如何使用 GetSecretValue。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [GetSecretValue](#)。

## 使用 SDK for Rust 的 Amazon SES API v2 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon SES API v2 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- [動作](#)
- [案例](#)

### 動作

#### CreateContact

下列程式碼範例示範如何使用 CreateContact。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateContact](#)。

## CreateContactList

下列程式碼範例示範如何使用 CreateContactList。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [CreateContactList](#)。

## CreateEmailIdentity

下列程式碼範例示範如何使用 CreateEmailIdentity。

## SDK for Rust

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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) ),
  },
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [CreateEmailIdentity](#)。

**CreateEmailTemplate**

下列程式碼範例示範如何使用 CreateEmailTemplate。

## SDK for Rust

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)),
    },
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [CreateEmailTemplate](#)。

## DeleteContactList

下列程式碼範例示範如何使用 DeleteContactList。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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}") ),
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DeleteContactList](#)。

## DeleteEmailIdentity

下列程式碼範例示範如何使用 DeleteEmailIdentity。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DeleteEmailIdentity](#)。

## DeleteEmailTemplate

下列程式碼範例示範如何使用 DeleteEmailTemplate。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [DeleteEmailTemplate](#)。

## GetEmailIdentity

下列程式碼範例示範如何使用 GetEmailIdentity。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

確定是否已驗證電子郵件地址。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [GetTopicAttributes](#)。

## ListContactLists

下列程式碼範例示範如何使用 ListContactLists。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListContactLists](#)。

## ListContacts

下列程式碼範例示範如何使用 ListContacts。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListContacts](#)。

## SendEmail

下列程式碼範例示範如何使用 SendEmail。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

向聯絡人列表中的所有成員傳送消息。

```
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(())
}
```

使用範本傳送訊息給聯絡人清單的所有成員。

```

        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)),
    }

```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [SendEmail](#)。

## 案例

### 電子報案例

下列程式碼範例示範如何執行 Amazon SES API v2 電子報案例。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的下列主題。

- [CreateContact](#)
- [CreateContactList](#)
- [CreateEmailIdentity](#)
- [CreateEmailTemplate](#)
- [DeleteContactList](#)

- [DeleteEmailIdentity](#)
- [DeleteEmailTemplate](#)
- [ListContacts](#)
- [SendEmail.simple](#)
- [SendEmail.template](#)

## 使用 SDK for Rust 的 Amazon SNS 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon SNS 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [動作](#)
- [案例](#)
- [無伺服器範例](#)

## 動作

### CreateTopic

下列程式碼範例示範如何使用 CreateTopic。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱適用於 Rust API 的 AWS SDK 參考中的 [CreateTopic](#)。

## ListTopics

下列程式碼範例示範如何使用 ListTopics。

SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [ListTopics](#)。

## Publish

下列程式碼範例示範如何使用 Publish。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的[發佈](#)。

## Subscribe

下列程式碼範例示範如何使用 Subscribe。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

訂閱主題的電子郵件地址。

```
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(())  
}
```

- 如需 API 詳細資訊，請參閱適用於 Rust API 的 AWS SDK 參考中的[訂閱](#)。

## 案例

### 建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

#### SDK for Rust

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

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

## 無伺服器範例

### 使用 Amazon SNS 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，以接收從 SNS 主題接收訊息所觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。



## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 SNS 事件。

```
// 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 record 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 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon SQS 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [動作](#)
- [無伺服器範例](#)

## 動作

### ListQueues

下列程式碼範例示範如何使用 ListQueues。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

擷取 區域中列出的第一個 Amazon SQS 佇列。

```
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())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ListQueues](#)。

## ReceiveMessage

下列程式碼範例示範如何使用 ReceiveMessage。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API 參考中的 [ReceiveMessage](#)。

## SendMessage

下列程式碼範例示範如何使用 SendMessage。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱 SDK AWS for Rust API 參考中的 [SendMessage](#)。

## 無伺服器範例

使用 Amazon SQS 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，以接收從 SQS 佇列接收訊息所觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。

## SDK for Rust

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 SQS 事件。

```
// 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
}
```

## 使用 Amazon SQS 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何為從 SQS 佇列接收事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 報告 SQS 批次項目失敗。

```
// 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 SDK for Rust 的範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 [來執行動作和實作常見案例 AWS STS](#)。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [動作](#)

### 動作

#### AssumeRole

下列程式碼範例示範如何使用 AssumeRole。

SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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),
}
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [AssumeRole](#)。

## 使用 SDK for Rust 的 Systems Manager 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Systems Manager 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

### 主題

- [動作](#)

## 動作

### DescribeParameters

下列程式碼範例示範如何使用 DescribeParameters。



## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [DescribeParameters](#)。

## GetParameter

下列程式碼範例示範如何使用 GetParameter。

## SDK for Rust

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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::
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [GetParameter](#)。

## PutParameter

下列程式碼範例示範如何使用 PutParameter。

### SDK for Rust

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [PutParameter](#)。

## 使用 SDK for Rust 的 Amazon Transcribe 範例

下列程式碼範例示範如何使用 AWS SDK for Rust 搭配 Amazon Transcribe 來執行動作和實作常見案例。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執程式碼的指示。

主題

- [案例](#)

### 案例

將文字轉換為語音然後返回文字

以下程式碼範例顯示做法：

- 使用 Amazon Polly 將純文字 (UTF-8) 輸入檔案合成至音訊檔案中。
- 將音訊檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Transcribe 將音訊檔案轉換為文字。
- 顯示文字。

SDK for Rust

使用 Amazon Polly 將純文字 (UTF-8) 輸入檔案合成至音訊檔案中，將音訊檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Transcribe 將該音訊檔案轉換為文字，然後顯示文字。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

## 此範例中使用的服務

- Amazon Polly
- Amazon S3
- Amazon Transcribe

# 此 AWS 產品或服務的安全性

雲端安全是 Amazon Web Services (AWS) 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。安全性是 AWS 和 之間的共同責任。[共同責任模型](#) 將此描述為雲端本身的安全和雲端內部的安全。

雲端的安全性 – AWS 負責保護執行在 AWS 雲端中提供的所有服務的基礎設施，並提供您可以安全使用的服務。我們的安全責任是 的最高優先順序 AWS，而且第三方稽核人員會定期測試和驗證我們的安全有效性，做為[AWS 合規計畫](#)的一部分。

雲端的安全性 – 您的責任取決於您使用 AWS 的服務，以及其他因素，包括資料的敏感度、組織的需求，以及適用的法律和法規。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循[共同責任模型](#)。如需 AWS 服務安全資訊，請參閱[AWS 服務安全文件頁面](#)，以及[AWS 合規計畫在 AWS 合規工作範圍內的服務](#)。

## 主題

- [此 AWS 產品或服務中的資料保護](#)
- [此 AWS 產品或服務的合規驗證](#)
- [此 AWS 產品或服務的基礎設施安全](#)
- [在中強制執行最低 TLS 版本 適用於 Rust 的 AWS SDK](#)

# 此 AWS 產品或服務中的資料保護

AWS [共同責任模型](#)適用於此 AWS 產品或服務中的資料保護。如此模型所述，AWS 負責保護執行所有的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱[資料隱私權常見問答集](#)。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 [AWS 共同的責任模型和 GDPR](#) 部落格文章。

基於資料保護目的，我們建議您保護 AWS 帳戶 登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。

- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《使用者指南》中的[使用 CloudTrail 追蹤](#)。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱[聯邦資訊處理標準 \(FIPS\) 140-3](#)。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用此 AWS 產品或服務，或使用主控台、API AWS CLI或 AWS SDKs的其他 AWS 服務 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

## 此 AWS 產品或服務的合規驗證

若要了解 是否 AWS 服務 在特定合規計劃的範圍內，請參閱[AWS 服務 合規計劃範圍內](#)然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 [AWS Compliance Programs](#)。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱[在 中下載報告 AWS Artifact](#)。

您在使用 時的合規責任 AWS 服務 取決於資料的敏感度、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源以協助合規：

- [安全合規與治理](#) - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。
- [HIPAA 合格服務參考](#) - 列出 HIPAA 合格服務。並非所有 AWS 服務 都符合 HIPAA 資格。
- [AWS 合規資源](#) - 此工作手冊和指南的集合可能適用於您的產業和位置。
- [AWS 客戶合規指南](#) - 透過合規的角度了解共同責任模型。本指南摘要說明保護 的最佳實務，AWS 服務 並將指南映射到跨多個架構的安全控制 (包括國家標準和技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO))。
- AWS Config 開發人員指南中的[使用規則評估資源](#) - AWS Config 服務會評估資源組態符合內部實務、產業準則和法規的程度。
- [AWS Security Hub](#) - 這 AWS 服務 可讓您全面檢視其中的安全狀態 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱「[Security Hub 控制參考](#)」。

- [Amazon GuardDuty](#) – 這可透過監控您的環境是否有可疑和惡意活動，來 AWS 服務偵測對您 AWS 帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- [AWS Audit Manager](#) – 這 AWS 服務可協助您持續稽核 AWS 用量，以簡化您管理風險的方式，以及符合法規和產業標準的方式。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務遵循[共同責任模型](#)。如需 AWS 服務安全資訊，請參閱[AWS 服務安全文件頁面](#)，以及[AWS 合規計劃在 AWS 合規工作範圍內的服務](#)。

## 此 AWS 產品或服務的基礎設施安全

此 AWS 產品或服務使用受管服務，因此受到全球網路安全的 AWS 保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的相關資訊，請參閱[AWS 雲端安全](#)。若要使用基礎設施安全的最佳實務設計您的 AWS 環境，請參閱 Security Pillar AWS Well-Architected Framework 中的[基礎設施保護](#)。

您可以使用 AWS 已發佈的 API 呼叫，透過網路存取此 AWS 產品或服務。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以透過[AWS Security Token Service](#) (AWS STS) 來產生暫時安全憑證來簽署請求。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循[共同責任模型](#)。如需 AWS 服務安全資訊，請參閱[AWS 服務安全文件頁面](#)，以及[AWS 合規計劃在 AWS 合規工作範圍內的服務](#)。

## 在中強制執行最低 TLS 版本 適用於 Rust 的 AWS SDK

適用於 Rust 的 AWS SDK 使用 TLS 來提高 AWS 與服務通訊時的安全性。軟體開發套件預設會強制執行最低 TLS 版本 1.2。根據預設，軟體開發套件也會交涉用戶端應用程式和服務可用的最高 TLS 版本。例如，軟體開發套件可能可以交涉 TLS 1.3。

特定 TLS 版本可在應用程式中強制執行，方法是提供 SDK 使用的 TCP 連接器手動組態。為了說明這一點，下列範例示範如何強制執行 TLS 1.3。

### Note

有些 AWS 服務尚未支援 TLS 1.3，因此強制執行此版本可能會影響 SDK 互通性。我們建議您在生產部署之前，先針對每個服務測試此組態。

```
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(())
}
```

## 附錄：開發套件使用的 Rust 板條箱

此附錄包含有關所使用的箱子的進階資訊，適用於 Rust 的 AWS SDK。這包括其使用的 Smithy 元件、在特定建置情況下可能需要使用的箱子，以及其他資訊。

### Smithy 條板箱

適用於 Rust 的 AWS SDK 是以 [Smithy](#) 為基礎，就像大多數的 AWS SDKs 一樣。Smithy 是一種語言，用於描述 SDK 提供的資料類型和函數。然後，這些模型會用來協助建置 SDK 本身。

查看適用於 Rust 的 SDK 板條箱及其 Smithy 相依性的版本時，知道這些板條箱都使用 [標準語意版本編號](#) 可能會有所幫助。

如需 Rust 的 Smithy 板條箱的其他詳細資訊，請參閱 [Smithy Rust 設計](#)。

### 與適用於 Rust 的 SDK 搭配使用的條板箱

由發佈了許多 Smithy 條板箱 AWS。其中有些與適用於 Rust 使用者的 SDK 相關，有些則是實作詳細資訊：

`aws-smithy-async`

如果您不使用 Tokio 進行非同步功能，請包含此條板箱。

`aws-smithy-runtime`

包括 AWS SDKs 所需的建置區塊。

`aws-smithy-runtime-api`

軟體開發套件所使用的基礎介面。

`aws-smithy-types`

從 AWS SDKs 重新匯出的類型。如果您使用多個 SDKs 請使用此選項。

`aws-smithy-types-convert`

用於移入和移出的公用程式函數 `aws-smithy-types`。

### 其他條板箱

存在下列條板箱，但您應該不需要知道這些條板箱的任何相關資訊：

適用於 Rust 的 SDK 使用者不需要的伺服器相關 Crate :

- `aws-smithy-http-server`
- `aws-smithy-http-server-python`

包含 SDK 使用者不需要使用的under-the-hood程式碼的條板箱 :

- `aws-smithy-checksum-callbacks`
- `aws-smithy-eventstream`
- `aws-smithy-http`
- `aws-smithy-protocol-test`
- `aws-smithy-query`
- `aws-smithy-json`
- `aws-smithy-xml`

不支援且未來會消失的條板箱 :

- `aws-smithy-client`
- `aws-smithy-http-auth`
- `aws-smithy-http-tower`

# 文件歷史紀錄

本主題說明 適用於 Rust 的 AWS SDK 開發人員指南在其歷史記錄過程中的重要變更。

變更	描述	日期
<a href="#">重組目錄</a>	重組目錄，以更妥善區分組態與用量。	2024 年 7 月 1 日
<a href="#">的一般可用性 適用於 Rust 的 AWS SDK</a>	更新指南，以包含新的安全資訊、新的和更新的程式碼範例、有關使用範例進行單元測試的新詳細資訊，以及 SDK 新通用版本的其他新和更新的內容。	2023 年 11 月 27 日
<a href="#">強制執行最低 TLS 版本</a>	新增如何在 SDK 中強制執行 TLS 版本的相關資訊。	2022 年 5 月 4 日
<a href="#">適用於 Rust 的 AWS SDK 開發人員預覽版本</a>	<a href="#">開發人員預覽版本</a>	2021 年 12 月 2 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。