Developer Guide

AWS SDK for Rust

AWS SDK for Rust: Developer Guide

Table of Contents

What is the AWS SDK for Rust?

Rust is a systems programming language without a garbage collector focused on three goals: safety, speed, and concurrency.

The AWS SDK for Rust (the SDK) provides Rust APIs to interact with Amazon Web Services infrastructure services. Using the SDK, you can build applications on top of Amazon S3, Amazon EC2, DynamoDB, and more.

Topics

- Getting started with the SDK
- Maintenance and support for SDK major versions
- Additional resources

Getting started with the SDK

If you are a first-time user of the SDK, we recommend that you begin by reading Get started with the AWS SDK for Rust.

For configuration and setup, including how to create and configure service clients for making requests to AWS services, see Configure the AWS SDK for Rust.

For information about using the SDK, see Use the AWS SDK for Rust.

For a complete list of Rust code examples, see Code examples.

Maintenance and support for SDK major versions

For information about maintenance and support for SDK major versions and their underlying dependencies, see the following in the AWS SDKs and Tools Reference Guide:

- AWS SDKs and Tools Maintenance Policy
- AWS SDKs and Tools Version Support Matrix

Additional resources

In addition to this guide, the following are valuable online resources for SDK developers:

- [AWS SDKs and Tools Reference Guide](): Contains settings, features, and other foundational concepts common among AWS SDKs.
- [Rust Programming Language web site]()
- [AWS SDK for RustAPI Reference]()
- [AWS Developer Tools Blog]()
- [AWS SDK for Rust source code]() on GitHub
- [The AWS Code Sample Catalog]()

Get started with the AWS SDK for Rust

Learn how to install, set up, and use the SDK to create a Rust application to access an AWS resource programmatically.

Topics

- [SDK authentication with AWS](#)

- [Hello tutorial for the AWS SDK for Rust](#)

- [Fundamentals](#)

SDK authentication with AWS

You must establish how your code authenticates with AWS when developing with AWS services. You can configure programmatic access to AWS resources in different ways depending on the environment and the AWS access available to you.

To choose your method of authentication and configure it for the SDK, see [Authentication and access](#) in the *AWS SDKs and Tools Reference Guide*.

We recommend that new users who are developing locally and are not given a method of authentication by their employer should set up AWS IAM Identity Center. This method includes installing the AWS CLI for ease of configuration and for regularly signing in to the AWS access portal. If you choose this method, your environment should contain the following elements after you complete the procedure for [IAM Identity Center authentication](#) in the *AWS SDKs and Tools Reference Guide*:

- The AWS CLI, which you use to start an AWS access portal session before you run your application.

- A [shared AWSconfig file](#) having a `[default]` profile with a set of configuration values that can be referenced from the SDK. To find the location of this file, see [Location of the shared files](#) in the *AWS SDKs and Tools Reference Guide*.

- The shared `config` file sets the [`region`](#) setting. This sets the default AWS Region that the SDK uses for AWS requests. This Region is used for SDK service requests that aren't specified with a Region to use.

- The SDK uses the profile's [SSO token provider configuration](#) to acquire credentials before sending requests to AWS. The `sso_role_name` value, which is an IAM role connected to an IAM Identity Center permission set, allows access to the AWS services used in your application.

  The following sample `config` file shows a default profile set up with SSO token provider configuration. The profile's `sso_session` setting refers to the named [sso-session section](#). The `sso-session` section contains settings to initiate an AWS access portal session.

```
[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
```

The SDK for Rust does not need additional packages (such as `SSO` and `SSOOIDC`) to be added to your application to use IAM Identity Center authentication.

Start an AWS access portal session

Before running an application that accesses AWS services, you need an active AWS access portal session for the SDK to use IAM Identity Center authentication to resolve credentials. Depending on your configured session lengths, your access will eventually expire and the SDK will encounter an authentication error. To sign in to the AWS access portal, run the following command in the AWS CLI.

```
$ aws sso login
```

If you followed the guidance and have a default profile setup, you do not need to call the command with a `--profile` option. If your SSO token provider configuration is using a named profile, the command is `aws sso login --profile named-profile`.

To optionally test if you already have an active session, run the following AWS CLI command.

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

If your session is active, the response to this command reports the IAM Identity Center account and permission set configured in the shared `config` file.

> **ⓘ Note**
>
> If you already have an active AWS access portal session and run `aws sso login`, you will not be required to provide credentials.
> The sign-in process might prompt you to allow the AWS CLI access to your data. Because the AWS CLI is built on top of the SDK for Python, permission messages might contain variations of the `botocore` name.

More authentication information

Human users, also known as *human identities*, are the people, administrators, developers, operators, and consumers of your applications. They must have an identity to access your AWS environments and applications. Human users that are members of your organization - that means you, the developer - are known as *workforce identities*.

Use temporary credentials when accessing AWS. You can use an identity provider for your human users to provide federated access to AWS accounts by assuming roles, which provide temporary credentials. For centralized access management, we recommend that you use AWS IAM Identity Center (IAM Identity Center) to manage access to your accounts and permissions within those accounts. For more alternatives, see the following:

- To learn more about best practices, see [Security best practices in IAM](#) in the *IAM User Guide*.
- To create short-term AWS credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*.
- To learn about other credential providers supported by SDK for Rust, see [Standardized credential providers](#) in the *AWS SDKs and Tools Reference Guide*.

Hello tutorial for the AWS SDK for Rust

Prerequisites

In order to use the AWS SDK for Rust, you must have Rust and Cargo installed.

- Install the Rust toolchain: https://www.rust-lang.org/tools/install
- Install the `cargo-component` tool by running command: `cargo install cargo-component`

Recommended tools:

The following optional tools can be installed in your IDE to assist with code completion and troubleshooting.

- The rust-analyzer extension, see Rust in Visual Studio Code.
- Amazon Q Developer, see Installing the Amazon Q Developer extension or plugin in your IDE.

Create your first SDK app

This procedure creates your first SDK for Rust application that lists your DynamoDB tables.

1. In a terminal or console window, navigate to a location on your computer where you want to create the app.

2. Run the following command to create a `hello_world` directory and populate it with a skeleton Rust project:

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

3. Navigate into the `hello_world` directory and use the following command to add the required dependencies to the app:

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

   These dependencies include the SDK crates that provide configuration features and support for DynamoDB, including the tokio crate, which is used to implement asynchronous I/O operations.

   > **ⓘ Note**
   >
   > Unless you use a feature like `tokio/full` Tokio will not provide an async runtime. The SDK for Rust requires an async runtime.

4. Update `main.rs` in the `src` directory to contain the following code.

```rust
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**
>
> This example only displays the first page of results. See [the section called "Pagination"](#)
> to learn how to handle multiple pages of results.

5.   Run the program:

```
$ cargo run
```

You should see a list of your table names.

Fundamentals

Prerequisites

In order to use the AWS SDK for Rust, you must have Rust and Cargo installed.

- Install the Rust toolchain: https://www.rust-lang.org/tools/install

- Install the `cargo-component` tool by running command: `cargo install cargo-component`

Recommended tools:

The following optional tools can be installed in your IDE to assist with code completion and troubleshooting.

- The rust-analyzer extension, see Rust in Visual Studio Code.

- Amazon Q Developer, see Installing the Amazon Q Developer extension or plugin in your IDE.

Rust fundamentals

The following are some basics of the Rust programming language that would be helpful to know. All references for more information come from The Rust Programming Language.

- `Cargo.toml` is the standard Rust project configuration file, it contains the dependencies and some metadata about the project. Rust source files have a `.rs` file extension. See Hello, Cargo!.

  - The `Cargo.toml` can be customized with profiles, see Customizing Builds with Release Profiles. These profiles are completely unrelated and independent from AWS's use of profiles within the shared AWS `config` file.

  - A common way to add library dependencies to your project and this file is to use `cargo add`. See `cargo-add`.

- Rust has a basic function structure like the following. The `let` keyword declares a variable and might be paired with assignment (=). If you don't specify a type after `let`, then the compiler will infer one. See Variables and Mutability.

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

- To declare a variable x with an explicit type T, Rust uses syntax x: T. See [Data Types](#).

- struct X {} defines the new type X. Methods are implemented on the custom struct type X. Methods for type X are declared with implementation blocks prefixed with keyword impl. Inside the implementation block, self refers to the instance of the struct that the method was called on. See [Keyword impl](#) and [Method Syntax](#).

- If an exclamation point ("!") follows what appears to be a function definition or function call, then the code is defining or calling a macro. See [Macros](#).

- In Rust, unrecoverable errors are represented by the panic! macro. When a program encounters a panic! it will stop running, print a failure message, unwind, clean up the stack, and quit. See [Unrecoverable Errors with panic!](#).

- Rust doesn't support inheritance of functionality from base classes like other programming languages do; traits are how Rust provides the overloading of methods. Traits might be thought of as being conceptually similar to an interface. However, traits and true interfaces have differences and are often used differently in the design process. See [Traits: Defining Shared Behavior](#).

  - Polymorphism refers to code that supports functionality for multiple data types without having to individually write each one. Rust supports polymorphism through enums, traits, and generics. See [Inheritance as a Type System and as Code Sharing](#).

- Rust is very explicit about memory. Smart pointers "are data structures that act like a pointer but also have additional metadata and capabilities". See [Smart Pointers](#).

  - The type Cow is a clone-on-write smart pointer that helps transfer memory ownership to the caller when necessary. See [Enum std::borrow::Cow](#).

  - The type Arc is a Atomically Reference Counted smart pointer that counts allocated instances. See [Struct std::sync::Arc](#).

- The SDK for Rust frequently uses the builder pattern for constructing complex types.

AWS SDK for Rust crate fundamentals

- The SDK for Rust library is separated into different library crates by each AWS service. These crates are available at https://docs.rs/.

- AWS service crates follow the naming convention of `aws-sdk-`*`[servicename]`*, such as `aws-sdk-s3` and `aws-sdk-dynamodb`.

- The primary core crate for the SDK for Rust functionality is `aws-config`. This is included in the majority of projects because it provides functionality to read configuration from the environment.

  - Don't confuse this with the AWS service that is called AWS Config. Since that is a service, it follows the standard convention and is called `aws-sdk-config`.

Project configuration for working with AWS services

- You will need to add a crate to your project for each AWS service that you want your application to use.

- The recommended way to add a crate is using the command line in your project's directory by running `cargo add ` *`[crateName]`*, such as `cargo add aws-sdk-s3`.

  - This will add a line to your project's `Cargo.toml` under `[dependencies]`.

  - By default, this will add the latest version of the crate to your project.

- In your source file, use the `use` statement to bring items from their crates into scope. See Using External Packages on the Rust Programming Language website.

  - Crate names are often hyphenated, but the hyphens get converted to underscores when actually using the crate. For example, the `aws-config` crate is used in code `use` statement as: `use aws_config`.

- Configuration is a complex topic. Configuration can occur directly in code, or be specified externally in environment variables or config files. For more information, see Configuration options.

  - When the SDK loads your configuration, invalid values are logged instead of halting execution because most settings have reasonable defaults. To learn how to turn on logging, see Enable logging of AWS SDK for Rust code.

  - Most environment variables and config file settings are loaded once when your program starts. Any updates to the values will not be seen until you restart your program.

Tokio runtime

- Tokio is an asynchronous runtime for the SDK for Rust programming language, it executes the async tasks. See tokio.rs and docs.rs/tokio.

- The SDK for Rust requires an async runtime. We recommend you add the following crate to your projects:

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

- The `tokio::main` attribute macro creates an async main entry point to your program. To use this macro, add it to the line before your `main` method, as shown in the following:

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

Configure the AWS SDK for Rust

Learn how to configure the AWS SDK for Rust. You must establish how your code authenticates with AWS when you develop with AWS services. You must also set the AWS Region you want to use.

The [AWS SDKs and Tools Reference Guide](#) also contains settings, features, and other foundational concepts common among many of the AWS SDKs.

Topics

- [Create a service client](#)

- [Behavior versions](#)

- [Credential Providers](#)

- [AWS Region selection](#)

- [Configuration options](#)

- [Client endpoints](#)

- [HTTP](#)

- [Interceptors](#)

- [Use observability features](#)

- [Override a single operation configuration of client](#)

- [Retries](#)

- [Timeouts](#)

Create a service client

To make a request to an AWS service, you first instantiate a client for that service. You can configure common settings for service clients such as timeouts, the HTTP client, and retry configuration.

Each service client requires an AWS Region and a credential provider. The SDK uses these values to send requests to the correct Region for your resources and to sign requests with the correct credentials. You can specify these values programmatically in code or have them automatically loaded from the environment.

> **ⓘ Note**
>
> Service clients can be expensive to construct and are generally meant to be shared. To facilitate this, all `Client` structs implement `Clone`.

The SDK has a series of places (or sources) that it checks in order to find a value for configuration settings.

1. Any explicit setting set in the code or on a service client itself takes precedence over anything else.

2. Environment variables

   - For details on setting environment variables, see [environment variables](#) in the *AWS SDKs and Tools Reference Guide*.

3. Shared `config` and `credentials` files

   - For details on setting up these files, see the [Shared `config` and `credentials` files](#) in the *AWS SDKs and Tools Reference Guide*.

4. Any default value provided by the SDK source code itself is used last.

   - Some properties, such as Region, don't have a default. You must specify them either explicitly in code, in an environment setting, or in the shared `config` file. If the SDK can't resolve required configuration, API requests can fail at runtime.

Most environment variable settings and the `config` and `credentials` file settings are shared by multiple AWS SDKs and tools for consistent behavior. To see all the all settings that the SDK can resolve from the environment variables or configuration files, see the [Settings reference](#) in the *AWS SDKs and Tools Reference Guide*.

Configure a client from the environment

To create a client with environment-sourced configuration, use static methods from the aws-config crate:

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

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

Creating a client this way is useful when running on Amazon Elastic Compute Cloud, AWS Lambda, or any other context where the configuration of a service client is available directly from the environment. This decouples your code from the environment that it's running in and makes it easier to deploy your application to multiple AWS Regions without changing the code.

You can explicitly override specific properties. Explicit configuration takes precedence over configuration resolved from the execution environment. The following example loads configuration from the environment, but explicitly overrides the AWS Region:

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

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

> **ⓘ Note**
>
> Not all configuration values are sourced by the client at creation time. Credential-related settings, such as temporary access keys and IAM Identity Center configuration, are accessed by the credential provider layer when the client is used to make a request.

The code `BehaviorVersion::latest()` shown in the prior examples indicates the version of the SDK to use for defaults. `BehaviorVersion::latest()` is appropriate for most cases. For details, see [Behavior versions](#).

Use the builder pattern for service-specific settings

There are some options that can only be configured on a specific service client type. However, most often, you'll still want to load the majority of configuration from the environment, and then specifically add the additional options. The builder pattern is a common pattern within the AWS SDK for Rust crates. You first load the general configuration using `aws_config::defaults`, then use the `from` method to load that configuration into the builder for the service you are working with. You can then set any unique configuration values for that service and call `build`. Lastly, the client is created from this modfied configuration.

```
// 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);
```

One way to discover additional methods that are available for a specific type of service client is to use the API documentation, such as for <u>aws_sdk_s3::config::Builder</u>.

Advanced explicit client configuration

To configure a service client with specific values instead of loading a configuration from the environment, you can specify them on the client `Config` builder as shown in the following:

```
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);
```

When you create a service configuration with `aws_sdk_s3::Config::builder()`, *no default configuration is loaded*. Defaults are only loaded when creating a config based on `aws_config::defaults`.

There are some options that can only be configured on a specific service client type. The previous example shows an example of this by using the `endpoint_resolver` function on a Amazon S3 client.

Behavior versions

AWS SDK for Rust developers expect and rely upon the robust and predictable behavior the language and its major libraries offer. To help developers using the SDK for Rust get the

expected behavior, client configurations are required to include a `BehaviorVersion`. The `BehaviorVersion` specifies the version of the SDK whose defaults are expected. This lets the SDK evolve over time, changing best practices to match new standards and support new features without unexpected adverse impact on your application's behavior.

> ⚠️ **Warning**
>
> If you try to configure the SDK or create a client without explicitly specifying a `BehaviorVersion`, the constructor will panic.

For example, imagine that a new version of the SDK is released with a new default retry policy. If your application uses a `BehaviorVersion` matching a previous version of the SDK, then that prior configuration is used instead of the new default configuration.

Each time a new behavior version of the SDK for Rust is released, the previous `BehaviorVersion` is marked with the SDK for Rust `deprecated` attribute and the new version is added. This causes warnings to occur at compile time, but otherwise lets the build continue as usual. `BehaviorVersion::latest()` is also updated to indicate the new version's default behavior.

> ℹ️ **Note**
>
> In most cases, you should use `BehaviorVersion::latest()` in code or the feature flag `behavior-version-latest` in the `Cargo.toml` file. It is recommended to pin to a specific version only as long as is required.

Set the behavior version in `Cargo.toml`

You can specify the behavior version for the SDK and individual modules,such as `aws-sdk-s3` or `aws-sdk-iam`, by including an appropriate feature flag in the `Cargo.toml` file. At this time, only the `latest` version of the SDK is supported in `Cargo.toml`:

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

Set the behavior version in code

Your code can change the behavior version as needed by specifying it when configuring the SDK or a client:

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

This example creates a configuration that uses the environment to configure the SDK but sets the `BehaviorVersion` to `v2023_11_09()`.

Credential Providers

To make requests to AWS using the AWS SDK for Rust, the SDK uses cryptographically-signed credentials issued by AWS. At runtime, the SDK retrieves configuration values for credentials by checking several locations.

If the retrieved configuration includes [AWS IAM Identity Center single sign-on access settings](), the SDK works with the IAM Identity Center to retrieve temporary credentials that it uses to make request to AWS services.

If the retrieved configuration includes [temporary credentials](), the SDK uses them to make AWS service calls. Temporary credentials consist of access keys and a session token.

Authentication with AWS can be handled outside of your codebase. Many authentication methods can be automatically detected, used, and refereshed by the SDK using the credential provider chain.

For guided options for getting started on AWS authentication for your project, see [Authentication and access]() in the *AWS SDKs and Tools Reference Guide*.

The credential provider chain

If you don't explicitly specify a credential provider when constructing a client, the SDK for Rust uses a credential provider chain that checks a series of places where you can supply credentials. Once the SDK finds credentials in one of these locations, the search stops. For details on constructing clients, see [Create a service client]().

The following example doesn't specify a credential provider in the code. The SDK uses the credential provider chain to detect the authentication that has been set up in the hosting environment, and uses that authentication for calls to AWS services.

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

Credential retrieval order

The credential provider chain searches for credentials using the following predefined sequence:

1. **Access key environment variables**

   The SDK attempts to load credentials from the AWS_ACCESS_KEY_ID and
   AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN environment variables.

2. **The shared AWS `config` and `credentials` files**

   The SDK attempts to load credentials from the [default] profile in the shared AWS config
   and credentials files. You can use the AWS_PROFILE environment variable to choose
   a named profile you want the SDK to load instead of using [default]. The config and
   credentials files are shared by various AWS SDKs and tools. For more information on these
   files, see the Shared config and credentials files in the *AWS SDKs and Tools Reference Guide*.

   If you use IAM Identity Center to authenticate, this is when the SDK for Rust uses the single sign-
   on token that was set up by running AWS CLI command aws sso login. The SDK uses the
   temporary credentials that the IAM Identity Center exchanged for a valid token. The SDK then
   uses the temporary credentials when it calls AWS services. For detailed information about this
   process, see Understand SDK credential resolution for AWS services in the *AWS SDKs and Tools
   Reference Guide*.

   - For guidance on configuring this provider, see IAM Identity Center authentication in the *AWS
     SDKs and Tools Reference Guide*.

   - For details on SDK configuration properties for this provider, see IAM Identity Center
     credential provider in the *AWS SDKs and Tools Reference Guide*.

3. **AWS STS web identity**

   When creating mobile applications or client-based web applications that require access to
   AWS, AWS Security Token Service (AWS STS) returns a set of temporary security credentials for
   federated users who are authenticated through a public identity provider (IdP).

   - When you specify this in a profile, the SDK or tool attempts to retrieve temporary credentials
     using AWS STS AssumeRoleWithWebIdentity API method. For details on this method, see
     AssumeRoleWithWebIdentity in the *AWS Security Token Service API Reference*.

- For guidance on configuring this provider, see Federate with web identity or OpenID Connect in the *AWS SDKs and Tools Reference Guide*.

- For details on SDK configuration properties for this provider, see Assume role credential provider in the *AWS SDKs and Tools Reference Guide*.

4. **Amazon ECS and Amazon EKS container credentials**

Your Amazon Elastic Container Service tasks and Kubernetes service accounts can have an IAM role associated with them. The permissions granted in the IAM role are assumed by the containers running in the task or containers of the pod. This role allows your SDK for Rust application code (on the container) to use other AWS services.

The SDK attempts to retrieve credentials from the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` or `AWS_CONTAINER_CREDENTIALS_FULL_URI` environment variables, which can be set automatically by Amazon ECS and Amazon EKS.

- For details on setting up this role for Amazon ECS, see Amazon ECS task IAM role in the *Amazon Elastic Container Service Developer Guide*.

- For Amazon EKS setup information, see Setting up the Amazon EKS Pod Identity Agent in the *Amazon EKS User Guide*.

- For details on SDK configuration properties for this provider, see Container credential provider in the *AWS SDKs and Tools Reference Guide*.

5. **Amazon EC2 Instance Metadata Service**

Create an IAM role and attach it to your instance. The SDK for Rust application on the instance attempts to retrieve the credentials provided by the role from the instance metadata.

- The SDK for Rust only supports IMDSv2.

- For details on setting up this role and using metadata, IAM roles for Amazon EC2 and Work with instance metadata in the *Amazon EC2 User Guide*.

- For details on SDK configuration properties for this provider, see IMDS credential provider in the *AWS SDKs and Tools Reference Guide*.

6. If credentials still aren't resolved at this point, the operation panics with an error.

For details on AWS credential provider configuration settings, see Standardized credential providers in the *Settings reference* of the *AWS SDKs and Tools Reference Guide*.

Explicit credential provider

Instead of relying on the credential provider chain to detect your authentication method, you can specify a specific credential provider that the SDK should use. When you load your general configuration using `aws_config::defaults`, you can specify a custom credential provider as shown in the following:

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

You can implement your own credential provider by implementing the [ProvideCredentials](#) trait.

Identity caching

The SDK will cache credentials and other identity types such as SSO tokens. By default, the SDK uses a lazy cache implementation that loads credentials upon first request, caches them, and then attempts to refresh them during another request when they are close to expiring. Clients created from the same `SdkConfig` will share an [IdentityCache](#).

AWS Region selection

You can access AWS services that operate in a specific geographic area by using AWS Regions. This can be useful both for redundancy and to keep your data and applications running close to where you and your users access them. For more information on how Regions are used, see [AWS Region](#) in the *AWS SDKs and Tools Reference Guide*.

> ⚠️ **Important**
>
>   Most resources reside in a specific AWS Region and you must supply the correct Region for the resource when using the SDK.

AWS Region provider chain

The following lookup process is used when loading a service client's configuration from the execution environment. The first value that the SDK finds set is used in the configuration of

the client. For more information on creating service clients, see Configure a client from the environment.

1. Any explicit Region set programmatically.

2. The `AWS_REGION` environment variable is checked.

   - If you are using the AWS Lambda service, this environment variable is set automatically by the AWS Lambda container.

3. The `region` property in the shared AWS `config` file is checked.

   - The `AWS_CONFIG_FILE` environment variable can be used to change the location of the shared `config` file. To learn more about where this file is kept, see Location of the shared `config and credentials files` in the *AWS SDKs and Tools Reference Guide*.

   - The `AWS_PROFILE` environment variable can be used to select a named profile instead of the default. To learn more about configuring different profiles, see Shared `config and credentials files` in the *AWS SDKs and Tools Reference Guide*.

4. The SDK attempts to use the Amazon EC2 Instance Metadata Service to determine the Region of the currently running Amazon EC2 instance.

   - The AWS SDK for Rust only supports IMDSv2.

Configuration options

Many configuration settings can be handled outside of your code. Most configuration settings can be set as either environment variables or in a separate shared AWS `config` file. The shared `config` file can maintain separate sets of settings, called profiles, to provide different configurations for different environments or tests.

Environment variables and shared `config` file settings are standardized and shared across AWS SDKs and tools to support consistent functionality across different languages.

See the *AWS SDKs and Tools Reference Guide* to learn about configuring your application through these methods, plus details on each cross-sdk setting. Some pages of interest might be:

- Shared `config and credentials files` - Explains profiles and the format of these configuration files.

- Location of the shared `config and credentials files` - Explains the default location of these files and how to change it.

- Environment variables support - Explains setting environment variables.

- [Setting reference](#) - Reference information on all cross-SDK settings.

Client endpoints

When the AWS SDK for Rust calls an AWS service, one of its first steps is to determine where to route the request. This process is known as endpoint resolution.

You can configure endpoint resolution for the SDK when you create a service client. The default configuration for endpoint resolution is usually fine, but there are several reasons why you might want to modify the default configuration. Two example reasons are as follows:

- To make requests to a pre-release version of a service or to a local deployment of a service.

- To access to specific service features not yet modeled in the SDK.

> ⚠️ **Warning**
>
> Endpoint resolution is an advanced SDK topic. If you change the default settings, you risk breaking your code. The default settings apply to most users in production environments.

Custom endpoints can be set globally so that they are used for all service requests, or you can set a custom endpoint for a specific AWS service.

Custom endpoints can be configured using environment variables or settings in the shared AWS `config` file. For information on this approach, see [Service-specific endpoints](#) in the *AWS SDKs and Tools Reference Guide*. For the complete list of shared `config` file settings and environment variables for all AWS services, see [Identifiers for service-specific endpoints](#).

Alternatively, this customization can also be configured in your code, as shown in the following sections.

Custom Configuration

You can customize endpoint resolution of a service client with two methods that are available when you build the client:

1. `endpoint_url(url: Into<String>)`

2. endpoint_resolver(resolver: impl crate::config::endpoint::ResolveEndpoint
   + `static)

You can set both properties. However, most often you provide only one. For general usage,
endpoint_url is most frequently customized.

Set endpoint URL

You can set a value for endpoint_url to indicate a "base" hostname for the service. This value,
however, is not final since it is passed as a parameter to the client's ResolveEndpoint instance.
The ResolveEndpoint implementation can then inspect and potentially modify that value to
determine the final endpoint.

Set endpoint resolver

A service client's ResolveEndpoint implementation determines the final resolved endpoint that
the SDK uses for any given request. A service client calls the resolve_endpoint method for
every request and uses the [EndpointFuture](#) value that is returned by the resolver with no further
changes.

The following example demonstrates supplying a custom endpoint resolver implementation for an
Amazon S3 client that resolves a different endpoint per-stage, such as staging and production:

```rust
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**
>
> Endpoint resolvers, and by extension the `ResolveEndpoint` trait, are specific to each
> service, and thus can only be configured on the service client config. Endpoint URL, on the
> other hand, can be configured either using shared config (applying to all services derived
> from it) or for a specific service.

ResolveEndpoint parameters

The `resolve_endpoint` method accepts service-specific parameters that contain properties used
in endpoint resolution.

Every service includes the following base properties:

Name	Type	Description
region	String	The client's AWS Region
endpoint	String	A string representation of the value set of `endpointUrl`
use_fips	Boolean	Whether FIPS endpoints are enabled in the client's configuration
use_dual_stack	Boolean	Whether dual-stack endpoints are enabled in the client's configuration

AWS services can specify additional properties required for resolution. For example, Amazon
S3 endpoint parameters includes the bucket name and also several Amazon S3-specific feature
settings. For example, the `force_path_style` property determines whether virtual host
addressing can be used.

If you implement your own provider, you shouldn't need to construct your own instance of endpoint parameters. The SDK provides the properties for each request and passes them to your implementation of `resolve_endpoint`.

Compare using `endpoint_url` to using `endpoint_resolver`

It is important to understand that the following two configurations, one that uses `endpoint_url` and the other that uses `endpoint_resolver`, do NOT produce clients with equivalent endpoint resolution behavior.

```
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();
```

The client that sets the `endpoint_url` specifies a *base* URL that is passed to the (default) provider, that can be modified as part of endpoint resolution.

The client that sets the `endpoint_resolver` specifies the *final* URL the Amazon S3 client uses.

Examples

Custom endpoints are often used for testing. Instead of making calls out to the cloud-based service, calls are routed to a locally-hosted, simulated service. Two such options are:

- DynamoDB local – a local version of the Amazon DynamoDB service.

- LocalStack – a cloud service emulator that runs in a container on your local machine.

The following examples illustrate two different ways to specify a custom endpoint to use these two testing options.

Use DynamoDB local directly in code

As described in the previous sections, you can set `endpoint_url` directly in code to override the base endpoint to point to the local DynamoDB server. In your code:

```
let config = aws_config::defaults(BehaviorVersion::latest())
     // Override the base endpoint to point to the local dynamodb server
    // which uses port 8000 by default
    .endpoint_url("http://localhost:8000")
    .load()
    .await;

let ddb = aws_sdk_dynamodb::Client::new(&config);
```

Use LocalStack using the `config` file

You can set service-specific endpoints in your shared AWS `config` file. The following configuration profile sets `endpoint_url` to connect to `localhost` on port 4566. For more information on LocalStack configuration, see Accessing LocalStack via the endpoint URL on the *LocalStack docs* website.

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

The SDK will pick up the changes in the shared `config` file and apply them to your SDK clients when you use the `localstack` profile. Using this approach, your code doesn't need to include any reference to endpoints, and would look like:

```
    // 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 = aws_sdk_s3::config::Builder::from(&config)
        .force_path_style(true)
        .build();
```

HTTP

The AWS SDK for Rust provides the following HTTP-related configurations:

Topics

- Hyper 1.x

Hyper 1.x

> **(i) Note**
>
> The AWS SDK for Rust was developed before `hyper` released a stable 1.0 version. The SDK
> currently defaults to using v0.14 version of `hyper`.

`hyper` is a lower-level HTTP library for Rust that can be used with the AWS SDK for Rust to make
API service calls. For the reference documentation on this crate, see hyper - Rust. By default, the
SDK for Rust uses v0.14 version of `hyper` and a `CryptoMode:Ring` provider.

Clients with `hyper` 1.0 support are located in the `aws-smithy-experimental` crate.

Choose a crypto provider

You must choose a `CryptoProvider` and enable the required feature in the `aws-smithy-experimental` crate.

The following `CryptoProvider` modes are available:

- `CryptoMode:AwsLc` - a cryptographic provider based on [aws-lc-rs](). This is a general-purpose cryptographic library maintained by the AWS Cryptography team for AWS and their customers, based on code from the Google BoringSSL project and the OpenSSL project.

  - Requires feature `crypto-aws-lc`.

- `CryptoMode:AwsLcFips` - a FIPS-compliant cryptographic provider based on [aws-lc-rs]().

  - Requires feature `crypto-aws-lc-fips`.

    > ⓘ **Note**
    >
    > This might require additional build tools to be able to compile. For more information, reference the [aws-lc-rs]() repository and build instructions.

- `CryptoMode:Ring` - a cryptographic provider based on [ring](). This is a "Safe, fast, small crypto using Rust".

  - Requires feature `crypto-ring`.

    > ⓘ **Note**
    >
    > This is the crypto provider that the SDK for Rust uses in the default `hyper` v0.14 client.

How to enable a feature for a crate

The following example shows how to add the `crypto-aws-lc` feature to the `aws-smithy-experimental` crate in your project's `Cargo.toml`:

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

Alternatively, you can run the following command at a command prompt in your project's folder:

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

For more information, see [Features]() in the *The Cargo Book*.

Create a hyper 1.0 client and use it with the SDK

To create a hyper 1.0 client, in your project's `Cargo.toml`, add the `aws-smithy-experimental` crate. The `HyperClientBuilder` requires you to choose a `CryptoProvider`. The previous section covered the provider choices and how to enable one of the crypto-related features of the `aws-smithy-experimental` crate.

The following code uses `CryptoMode::AwsLc` as the crypto provider for a hyper 1.0 HTTP client to use with an Amazon Simple Storage Service (Amazon S3) service client. For this example, your project's `Cargo.toml` also needs to include the AWS service's crate, `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);
```

The HTTP client can be set on the shared config struct or on a specific service's config struct.

Interceptors

You can use interceptors to hook into the execution of API requests and responses. Interceptors are open-ended mechanisms in which the SDK calls code that you write to inject behavior into the request/response lifecycle. This way, you can modify an in-flight request, debug request processing, view errors, and more.

The following example shows a simple interceptor that adds an additional header to all outgoing requests before the retry loop is entered:

```
use std::borrow::Cow;
use aws_smithy_runtime_api::client::interceptors::{
```

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

For more information and the available interceptor hooks, see the [Intercept](#) trait.

Interceptor registration

You register interceptors when you construct a service client or when you override configuration for a specific operation. The registration in different depending on whether you want the interceptor to apply to all operations for your client or only specific ones.

Interceptor for all operations on a service client

To register an interceptor for the entire client, add the interceptor using the `Builder` pattern.

```
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);
```

Interceptor for only a specific operation

To register an interceptor for only a single operation, use the `customize` extension. You can override your service client configurations at the per-operation level using this method. For more information on customizable operations, see [Override a single operation configuration of client](#).

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

Use observability features

Observability is the extent to which a system's current state can be inferred from the data it emits. The data emitted is commonly referred to as telemetry.

Topics

- [Enable logging of AWS SDK for Rust code](#)

Enable logging of AWS SDK for Rust code

The AWS SDK for Rust uses the [tracing](#) framework for logging.

1. In a command prompt for your project directory, add the [tracing-subscriber](#) crate as a dependency:

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

   This adds the crate to the [dependencies] section of your Cargo.toml file.

2. Initialize the subscriber. Usually this is done early in the main function before calling any SDK for Rust operation:

   ```
   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. Turn on logging using RUST_LOG environment variable. To enable the display of logging information, in a command prompt, set the RUST_LOG environment variable to the level you want to log at. The following example sets logging to the debug level:

Linux/macOS

```
$ RUST_LOG=debug
```

Windows

If you're using VSCode, the terminal window often defaults to PowerShell. Verify the type of prompt you are using.

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

PowerShell

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

4. Run the program:

```
$ cargo run
```

You should see additional output in the console or terminal window.

For more information see [filtering events with environment variables](#) from the `tracing-subscriber` documentation.

Interpret the log output

Once you have turned on logging by following the steps in the previous section, additional log information will be printed to standard out by default.

If you're using the default log output format (called "full" by the tracing module), the information that you see in the log output looks similar to this:

```
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
```

Each entry includes the following:

- The log entry's timestamp.

- The log level of the entry. This is a word such as INFO, DEBUG, or TRACE.

- The nested set of spans from which the log entry was generated, separated by colons (":"). This helps you identify the source of the log entry.

- The Rust module path containing the code that generated the log entry.

- The log message text.

The tracing module's standard output formats use ANSI escape codes to colorize the output. Keep these escape sequences in mind when filtering or searching the output.

> ⓘ **Note**
>
> The sdk_invocation_id that appears within the nested set of spans is a unique ID generated client side by the SDK to help correlate log messages. It is unrelated to the request ID found in the response from an AWS service.

Fine-tune your logging levels

If you use a crate that supports an environment filtering, such as tracing_subscriber, you can control the verbosity of logs by module.

You can turn on the same logging level for every module. The following sets the logging level to trace for every module:

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

You can turn on trace-level logging for a specific module. In the following example, only logs coming from `aws_smithy_runtime` will come in at `trace` level.

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

You can specify a different log level for multiple modules by separating them with commas. The following example sets the `aws_config` module to `trace` level logging, and the `aws_smithy_runtime` module to debug level logging.

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

The following table describes some of the modules that you can use to filter log messages:

Prefix	Description
aws_smithy_runtime	Request and response wire logging
aws_config	Credentials loading
aws_sigv4	Request signing and canonical requests

One way to figure out which modules you need to include in your log output is to first log everything, then find the crate name in the log output for the information you need. Then you can set the environment variable accordingly and run your program again.

Override a single operation configuration of client

After you [create a service client](#), configuration becomes immutable and will apply to all subsequent operations. While configuration can't be modified at this point, it can be overridden on a per-operation basis.

Each operation builder has a `customize` method available to create a `CustomizableOperation` so that you can override an individual copy of the existing configuration. The original client configuration will remain unmodified.

The following example shows the creation of an Amazon S3 client that calls two operations, the second of which is overridden to send to a different AWS Region. All of Amazon S3's object invocations use the `us-east-1` region except for when the API call is explicitly overridden to use the modified `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**
>
> The previous example is for Amazon S3, however the concept is the same for all operations. Certain operations might have additional methods on `CustomizeableOperation`.

For an example of adding an interceptor using `customize` for a single operation, see [Interceptor for only a specific operation](#).

Retries

The AWS SDK for Rust provides a default retry behavior and customizable configuration options. Calls to AWS services occasionally return unexpected exceptions. Certain types of errors, such as throttling or transient errors, might be successful if the call is retried.

Retry behavior can be configured globally using environment variables or settings in the shared AWS `config` file. For information on this approach, see [Retry behavior](#) in the *AWS SDKs and Tools Reference Guide*. It also includes detailed information on retry strategy implementations and how to choose one over another.

Alternatively, these options can also be configured in your code, as shown in the following sections.

Default retry configuration

Every service client defaults to the `standard` retry strategy configuration provided through the [`RetryConfig`](#) struct. By default, a call will be tried three times (*the initial attempt, plus two retries*). Additionally, each retry will be delayed by a short, random duration to avoid retry storms. This convention is suitable for the majority of use cases but might be unsuitable in specific circumstances such as high-throughput systems.

Only some types of errors are considered retryable by the SDKs. Examples of retryable errors are:

- socket timeouts
- service-side throttling
- transient service errors like HTTP 5XX responses

The following examples are **not** considered retryable:

- Missing or invalid parameters
- authentication/security errors
- misconfiguration exceptions

You can customize the `standard` retry strategy by setting the maximum attempts, delays, and backoff configuration.

Maximum Attempts

You can customize the maximum attempts in your code by supplying a modified [RetryConfig](#) to your `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;
```

Delays and backoff

If a retry is necessary, the default retry strategy waits before it makes the subsequent attempt. The delay for the first retry is small but it grows exponentially for later retries. The maximum amount of delay is capped so that it does not grow too large.

Random jitter is applied to the delays between all attempts. The jitter helps mitigate the effect of large fleets that can cause retry storms. For a deeper discussion about exponential backoff and jitter, see [Exponential Backoff And Jitter](#) in the *AWS Architecture Blog*.

You can customize the delay settings in your code by supplying a modified [RetryConfig](#) to your `aws_config::defaults`. The following code sets the configuration to delay the first retry attempt for up to 100 milliseconds and that the maximum amount of time between any retry attempt is 5 seconds.

```
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;
```

Adaptive Retry Mode

As an alternative to the `standard` mode retry strategy, the `adaptive` mode retry strategy is an advanced approach that seeks the ideal request rate to minimize throttling errors.

> ⓘ **Note**
>
> Adaptive retries is an advanced retry mode. Using this strategy is typically not recommended. See Retry behavior in the *AWS SDKs and Tools Reference Guide*.

Adaptive retries includes all the features of standard retries. It adds a client-side rate limiter that measures the rate of throttled requests compared to non-throttled requests. It also limits traffic to attempt to stay within a safe bandwidth, ideally causing zero throttling errors.

The rate adapts in real time to changing service conditions and traffic patterns and might increase or decrease the rate of traffic accordingly. Critically, the rate limiter might delay initial attempts in high-traffic scenarios.

You can select the `adaptive` retry strategy in code by supplying a modified RetryConfig:

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

Timeouts

The AWS SDK for Rust provides several settings for managing request timeouts and stalled data streams. These help your application behave optimally when unexpected delays and failures occur in the network.

API timeouts

When there are transient issues that could cause request attempts to take a long time or fail completely, it is important to review and set timeouts so that your application can fail fast and

behave optimally. Requests that fail can be automatically retried by the SDK. It is a good practice to set timeouts for both the individual attempt and the entire request.

The SDK for Rust provides a default timeout for establishing a connection for a request. The SDK doesn't have any default maximum wait-time set for recieving a response for a request attempt or for the entire request. The following timeout options are available:

Parameter	Default value	Description
Connect timeout	3.1 seconds	The maximum amount of time to wait to establish a connection before giving up.
Operation timeout	None	The maximum amount of time to wait before receiving a response from the SDK for Rust, including all retries.
Operation attempt timeout	None	The maximum amount of time to wait for a single HTTP attempt, after which the API call can be retried.
Read timeout	None	The maximum amount of time to wait to read the first byte of a response from the time the request is initiated.

The following example shows the configuration of an Amazon S3 client with custom timeout values:

```
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);
```

When you use both operation and attempt timeouts together, you set a hard limit on the total time spent on all attempts across retries. You also set an individual HTTP request to fail fast on a slow request.

As an alternative to setting these timeout values on the service client for all operations, you can configure or [override them for a single request](#) .

> ⚠️ **Important**
>
> Operation and attempt timeouts do not apply to streaming data consumed after the SDK for Rust has returned a response. As an example, consuming data from a `ByteStream` member of a response is not subject to operation timeouts.

Stalled stream protection

The SDK for Rust provides another form of timeout related to detecting stalled streams. A stalled stream is an upload or download stream that produces no data for longer than a configured grace period. This helps prevent applications from hanging indefinitely and never making progress.

Stalled stream protection will return an error when a stream is idle for longer than the acceptable period.

By default, the SDK for Rust enables stalled stream protection for both uploads and downloads and looks for at least 1 byte/sec of activity with a generous grace period of 20 seconds.

The following example shows a customized `StalledStreamProtectionConfig` that disables upload protection and changes the grace period for no activity to 10 seconds:

```
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**
>
> Stalled stream protection is an advanced configuration option. We recommend altering these values only if your application needs tighter performance or if it is causing some other issue.

Disable stalled stream protection

The following example shows how to disable stalled stream protection completely:

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

Use the AWS SDK for Rust

Learn common and recommended ways of using the AWS SDK for Rust to work with AWS services.

Topics

- Make requests
- Best practices
- Concurrency
- Error handling
- Create Lambda functions
- Pagination
- Create presigned URLs
- Unit testing
- Waiters

Make requests

To make a request to an AWS service, you must first create a service client. For each AWS service your code uses, it has its own crate and its own `Client` for interacting with it.

The `Client` exposes one method for each API operation exposed by the service. The return value of each of these methods is a "fluent builder", where different inputs for that API are added by builder-style function call chaining. After calling the service's methods, call `send()` to get a `Future` that will result in either a successful output or a `SdkError`. For more information on `SdkError`, see Error handling.

The following example demonstrates a basic operation using Amazon S3 to create a bucket in the `us-west-2` AWS Region:

```
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;
```

Each service crate has additional modules used for API inputs, such as the following:

- The `types` module has structs or enums to provide more complex structured information.
- The `primitives` module has simpler types for representing data such as date times or binary blobs.

See the [API reference documentation](#) for the service crate for more detailed crate organization and information. For example, the `aws-sdk-s3` crate for the Amazon Simple Storage Service has several [Modules](#). Two of which are:

- [aws_sdk_s3::types](#)
- [aws_sdk_s3::primitives](#)

Best practices

The following are best practices for using the AWS SDK for Rust.

Reuse SDK clients when possible

Depending on how an SDK client is constructed, creating a new client may result in each client maintaining its own HTTP connection pools, identity caches, and so on. We recommend sharing a client or at least sharing `SdkConfig` to avoid the overhead of expensive resource creation. All SDK clients implement `Clone` as a single atomic reference count update.

Configure API timeouts

The SDK provides default values for some timeout options, such as connection timeout and socket timeouts, but not for API call timeouts or individual API call attempts. It is a good practice to set

timeouts for both the individual attempt and the entire request. This will ensure your application fails fast in an optimal way when there are transient issues that could cause request attempts to take longer to complete or fatal network issues.

For more information on configuring operation timeouts, see [Timeouts](#).

Concurrency

The AWS SDK for Rust doesn't provide concurrency control but users have many options for implementing their own.

Terms

Terms related to this subject are easy to confuse and some terms have become synonyms even though they originally represented separate concepts. In this guide, we'll define the following:

- **Task**: Some "unit of work" that your program will run to completion, or attempt to run to completion.

  - **Sequential Computing**: When several tasks are executed one after another.

  - **Concurrent Computing**: When several tasks are executed in overlapping time periods.

  - **Concurrency**: The ability of a computer to complete multiple tasks in an arbitrary order.

  - **Multitasking**: The ability of a computer to run several tasks concurrently.

  - **Race Condition**: When the behavior of your program changes based on when a task is started or how long it takes to process a task.

  - **Contention**: Conflict over access to a shared resource. When two or more tasks want to access a resource at the same time, that resource is "in contention".

  - **Deadlock**: A state in which no more progress can be made. This typically happens because two tasks want to acquire each other's resources but neither task will release their resource until the other's resource becomes available. Deadlocks lead to a program becoming partly or completely unresponsive.

A simple example

Our first example is a sequential program. In later examples, we'll change this code using concurrency techniques. Later examples reuse the same

build_client_and_list_objects_to_download() method and make changes within
main().

The following example task is to download all the files in an Amazon Simple Storage Service
bucket:

1. Start by listing all the files. Save the keys in a list.

2. Iterate over the list, downloading each file in turn

```rust
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)
}
```

```rust
#[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**
>
> In these examples, we won't be handling errors, and we assume that the example bucket
> has no objects with keys that look like file paths. Thus, we won't cover creating nested
> directories.

Because of the architecture of modern computers, we can rewrite this program to be much more
efficient. We'll do that in a later example, but first, let's learn a few more concepts.

Ownership and mutability

Each value in Rust has a single owner. When an owner goes out of scope, all values it owns will
also be dropped. The owner can provide either one or more immutable references to a value **or** a
single mutable reference. The Rust compiler is responsible for ensuring that no reference outlives
its owner.

Additional planning and design is needed when multiple tasks need to mutably access the same
resource. In sequential computing, each task can mutably access the same resource without
contention because they run one after another in a sequence. However, in concurrent computing,
tasks can run in any order, and at the same time. Therefore, we must do more to prove to the
compiler that multiple mutable references are impossible (or at least to crash if they do occur).

The Rust standard library provides many tools to help us accomplish this. For more information on
these topics, see Variables and Mutability and Understanding Ownership in The Rust Programming
Language book.

More terms!

The following are lists of "synchronization objects". Altogether, they are the tools necessary to
convince the compiler that our concurrent program won't break ownership rules.

Standard library synchronization objects:

- **Arc**: An *Atomically Reference-Counted* pointer. When data is wrapped in an `Arc`, it can be shared freely, without worrying about any specific owner dropping the value early. In this sense, the ownership of the value becomes "shared". Values within an `Arc` cannot be mutable, but might have interior mutability.

- **Barrier**: Ensures multiple threads will wait for each other to reach a point in the program, before continuing execution all together.

- **Condvar**: a *Condition Variable* providing the ability to block a thread while waiting for an event to occur.

- **Mutex**: a *Mutual Exclusion* mechanism that ensures that at most one thread at a time is able to access some data. Generally speaking, a `Mutex` lock should never be held across an `.await` point in the code.

Tokio synchronization objects:

While the AWS SDKs are intended to be `async-runtime-agnostic`, we recommend the use of `tokio` synchronization objects for specific cases.

- **Mutex**: Similar to the standard library's `Mutex`, but with a slightly higher cost. Unlike the standard `Mutex`, this one can be held across an `.await` point in the code.

- **Sempahore**: A variable used to control access to a common resource by multiple tasks.

Rewriting our example to be more efficient (single-threaded concurrency)

In the following modified example, we use `futures_util::future::join_all` to run **ALL** `get_object` requests concurrently.

```
#[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;
}
```

This is the simplest way to benefit from concurrency, but it also has a few issues that might not be obvious at first glance:

1. We create all the request inputs at the same time. If we don't have enough memory to hold all the `get_object` request inputs then we'll run into an "out-of-memory" allocation error.

2. We create and await all the futures at the same time. Amazon S3 throttles requests if we try to download too much at once.

To fix both of these issues, we must limit the amount of requests that we're sending at any one time. We'll do this with a `tokio` [semaphore](#):

```
const CONCURRENCY_LIMIT: usize = 50;

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

    let get_object_futures = objects_to_download.into_iter().map(|object| {
        // Since each future needs to acquire a permit, we need to clone
        // the Arc'd semaphore before passing it in.
```

```
        let semaphore = concurrency_semaphore.clone();
        // We also need to clone the client so each task has its own handle.
        let client = client.clone();
        async move {
            let permit = semaphore
                .acquire()
                .await
                .expect("we'll get a permit if we wait long enough");
            let res = client
                .get_object()
                .key(&object)
                .bucket(EXAMPLE_BUCKET)
                .send()
                .await
                .expect("get_object succeeds");
            let body = res.body.collect().await.expect("body succeeds").into_bytes();
            tokio::fs::write(object, body).await.expect("write succeeds");
            std::mem::drop(permit);
        }
    });

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

We've fixed the potential memory usage issue by moving the request creation into the `async`
block. This way, requests won't be created until it's time to send them.

> **ⓘ Note**
>
> If you have the memory for it, it might be more efficient to create all your request inputs
> at once and hold them in memory until they're ready to be sent. To try this, move request
> input creation outside of the `async` block.

We've also fixed the issue of sending too many requests at once by limiting requests in flight to
`CONCURRENCY_LIMIT`.

> **ⓘ Note**
>
> The right value for `CONCURRENCY_LIMIT` is different for every project. When constructing
> and sending your own requests, try to set it as high as you can without running into

throttling errors. While it's possible to dynamically update your concurrency limit based on the ratio of successful to throttled responses that a service sends back, that's outside the scope of this guide due to its complexity.

Rewriting our example to be more efficient (multi-threaded concurrency)

In the previous two examples, we performed our requests concurrently. While this is more efficient than running them synchronously, we can make things even more efficient by using multi-threading. To do this with `tokio`, we'll need to spawn them as separate tasks.

> ⓘ **Note**
>
> This example requires that you use the multi-threaded `tokio` runtime. This runtime is gated behind the `rt-multi-thread` feature. And, of course, you'll need to run your program on a multi-core machine.

```
// 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;
}
```

Dividing work into tasks can be complex. Doing I/O (*input/output*) is typically blocking. Runtimes might struggle to balance the needs of long-running tasks with those of short-running tasks. Whatever runtime you choose, be sure to read their recommendations for the most efficient way to divide your work into tasks. For the `tokio` runtime recommendations, see Module `tokio::task`.

Debugging multi-threaded apps

Tasks running concurrently can be run in any order. As such, the logs of concurrent programs can very difficult to read. In the SDK for Rust, we recommend using the `tracing` logging system. It can group logs with their specific tasks, no matter when they're running. For guidance, see Enable logging of AWS SDK for Rust code.

A very useful tool for identifying tasks that have locked up is `tokio-console`, which is a diagnostic and debugging tool for asynchronous Rust programs. By instrumenting and running your program, and then running the `tokio-console` app, you can see a live view of the tasks your program is running. This view includes helpful information like the amount of time a task has spent waiting to acquire shared resources or the amount of times it has been polled.

Error handling

Understanding how and when the AWS SDK for Rust returns errors is important to building high-quality applications using the SDK. The following sections describe the different errors you might encounter from the SDK and how to handle them appropriately.

Every operation returns a `Result` type with the error type set to [SdkError<E, R = HttpResponse>](). SdkError is an enum with several possible types, called variants.

Service errors

The most common type of error is [SdkError::ServiceError](). This error represents an error response from an AWS service. For example, if you try to get an object from Amazon S3 that doesn't exist, Amazon S3 returns an error response.

When you encounter an `SdkError::ServiceError` it means that your request was successfully sent to the AWS service but could not be processed. This can be because of errors in the request's parameters or because of issues on the service side.

The error response details are included in the error variant. The following example shows how to conveniently get at the underlying `ServiceError` variant and handle different error cases:

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

Error metadata

Every service error has additional metadata that can be accessed by importing service-specific traits.

- The `<service>::error::ProvideErrorMetadata` trait provides access to any available underlying raw error code and error message returned from the service.

  - For Amazon S3, this trait is [aws_sdk_s3::error::ProvideErrorMetadata](#).

You can also get information that might be useful when troubleshooting service errors:

- The `<service>::operation::RequestId` trait adds extension methods to retrieve the unique AWS request ID that was generated by the service.

  - For Amazon S3, this trait is [aws_sdk_s3::operation::RequestId](#).

- The `<service>::operation::RequestIdExt` trait adds the `extended_request_id()` method to get an additional, extended request ID.

  - Only supported by some services.

  - For Amazon S3, this trait is [aws_sdk_s3::operation::RequestIdExt](#).

Detailed error printing with `DisplayErrorContext`

Errors in the SDK are generally the result of a chain of failures such as:

1. Dispatching a request has failed because the connector returned an error.

2. The connector returned an error because the credentials provider returned an error.

3. The credentials provider returned an error because it called a service and that service returned an error.

4. The service returned an error because the credentials request didn't have the correct authorization.

By default, display of this error only outputs "dispatch failure". This lacks details that help troubleshoot the error. The SDK for Rust provides a simple error reporter called `DisplayErrorContext`.

- The `<service>::error::DisplayErrorContext` struct adds functionality to output the full error context.

  - For Amazon S3, this struct is aws_sdk_s3::error::DisplayErrorContext.

When we wrap the error to be displayed and print it, `DisplayErrorContext` provides a much more detailed message similar to the following:

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

```
                                                        "content-type": HeaderValue { _private:
 H0("application/json") },

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

                                                        "access-control-expose-headers":
 HeaderValue { _private: H0("RequestId") },

                                                        "access-control-expose-headers":
 HeaderValue { _private: H0("x-amzn-RequestId") },

                                                        "requestid": HeaderValue { _private:
 H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") },

                                                        "server": HeaderValue { _private:
 H0("AWS SSO") },

                                                        "x-amzn-requestid": HeaderValue
 { _private: H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") }
                                                    }
                                                },
                                                body: SdkBody {
                                                    inner: Once(
                                                        Some(
                                                            b"{
                                                                \"message\":\"Session token not
 found or invalid\",

                                                                \"__type\":
\"com.amazonaws.switchboard.portal#UnauthorizedException\"}"
                                                        )
                                                    ),
                                                    retryable: true
                                                },
                                                extensions: Extensions {
                                                    extensions_02x: Extensions,
                                                    extensions_1x: Extensions
                                                }
                                            }
                                        }
                                    )
                                }
                            )
                        }
                    ),
                    connection: Unknown
                }
            }
)
```

Create Lambda functions

For detailed documentation on developing AWS Lambda functions with the AWS SDK for Rust, see [Building Lambda functions with Rust](#) in the *AWS Lambda Developer Guide*. That documentation guides you through using:

- The Rust Lambda runtime client crate for core functionality, `aws-lambda-rust-runtime`.
- The recommended command-line tool for deploying the Rust function binary to Lambda with [Cargo Lambda](#).

In addition to the guided examples that are in the *AWS Lambda Developer Guide*, there are also Lambda calculator example available in the [AWS SDK Code Examples Repository](#) on GitHub.

Pagination

Many AWS operations return truncated results when the payload is too large to return in a single response. Instead, the service returns a portion of the data and a token to retrieve the next set of items. This pattern is known as pagination.

The AWS SDK for Rust includes extension methods `into_paginator` on operation builders that can be used to automatically paginate the results for you. You only have to write the code that processes the results. All pagination operation builders have an `into_paginator()` method available that exposes a [`PaginationStream<Item>`](#) to paginate over the results.

- In Amazon S3, one example of this is
  [`aws_sdk_s3::operation::list_objects_v2::builders::ListObjectsV2FluentBuilder::`](#)

The following examples use Amazon Simple Storage Service. However, the concepts are the same for any service that has one or more paginated APIs.

The following code example shows the simplest example that uses the [`try_collect()`](#) method to collect all paginated results into a `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<_>>();
```

Sometimes, you want to have more control over paging and not pull everything into memory all at once. The following example iterates over objects in an Amazon S3 bucket until there are no more.

```
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);
    }
}
```

Create presigned URLs

You can presign requests for some AWS API operations so that another caller can use the request later without presenting their own credentials.

For example, assume that Jane has access to an Amazon Simple Storage Service (Amazon S3) object and she wants to temporarily share object access with Alejandro. Jane can generate a presigned `GetObject` request to share with Alejandro so that he can download the object without requiring access to Jane's credentials or having any of his own. The credentials used by the presigned URL are Jane's because she is the AWS user who generated the URL.

To learn more about presigned URLs in Amazon S3, see [Working with presigned URLs](#) in the *Amazon Simple Storage Service User Guide*.

Presigning basics

The AWS SDK for Rust provides a `presigned()` method on operation fluent-builders that can be used to get a presigned request.

The following example creates a presigned `GetObject` request for Amazon S3. The request is valid for 5 minutes after creation.

```
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?;
```

The `presigned()` method returns a `Result<PresignedRequest, SdkError<E, R>>`.

The returned `PresignedRequest` contains methods to get at the components of an HTTP request including the method, URI, and any headers. All of these need to be sent to the service, if present, for the request to be valid. Many presigned requests can be represented by the URI alone though.

Presigning POST and PUT requests

Many operations that are presignable require only a URL and must be sent as HTTP GET requests. Some operations, however, take a body and must be sent as an HTTP POST or HTTP PUT request along with headers in some cases. Presigning these requests is identical to presigning GET requests, but invoking the presigned request is more complicated.

The following is an example of presigning an Amazon S3 `PutObject` request and converting it into an `http::request::Request` which can be sent using an HTTP client of your choosing.

To use the `into_http_1x_request()` method, add the `http-1x` feature to your `aws-sdk-s3` crate in your `Cargo.toml` file:

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

Source file:

```
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);
```

Standalone Signer

> **ⓘ Note**
>
> This is an advanced use case. It isn't needed or recommended for most users.

There are a few use cases where it is necessary to create a signed request outside of the SDK for Rust context. For that you can use the `aws-sigv4` crate independently from the SDK.

The following is an example to demonstrate the basic elements, see the crate documentation for more details.

Add the `aws-sigv4` and `http` crates to your `Cargo.toml` file:

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

Source file:

```
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);
```

Unit testing

While there are many ways you can implement unit testing in your AWS SDK for Rust project, there are a few that we recommend:

- Use automock from the mockall crate to create and execute your tests.

- Use the AWS Smithy runtime's StaticReplayClient to create a fake HTTP client that can be used instead of the standard HTTP client that is normally used by AWS services. This client returns the HTTP responses that you specify rather than communicating with the service over the network, so that tests get known data for testing purposes.

Automatically generate mocks using mockall

You can automatically generate the majority of the mock implementations that your tests need by using the popular automock from the mockall crate .

This example tests a custom method called determine_prefix_file_size(). This method calls a custom list_objects() wrapper method that calls Amazon S3. By mocking list_objects(), the determine_prefix_file_size() method can be tested without actually contacting Amazon S3.

1. In a command prompt for your project directory, add the mockall crate as a dependency:

    ```
    $ cargo add mockall
    ```

    This adds the crate to the [dependencies] section of your Cargo.toml file.

2. Include the automock module from the mockall crate.

    Also include any other libraries related to the AWS service that you are testing, in this case, Amazon S3.

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

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

3. Next, add code that determines which of two implementation of the application's Amazon S3 wrapper structure to use.

    -   The real one written to access Amazon S3 over the network.

    -   The mock implementation generated by `mockall`.

    In this example, the one that's selected is given the name S3. The selection is conditional based on the `test` attribute:

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

4. The S3Impl struct is the implementation of the Amazon S3 wrapper structure that actually sends requests to AWS.

    -   When testing is enabled, this code isn't used because the request is sent to the mock and not AWS. The dead_code attribute tells the linter not to report a problem if the S3Impl type isn't used.

    -   The conditional #[cfg_attr(test, automock)] indicates that when testing is enabled, the automock attribute should be set. This tells `mockall` to generate a mock of S3Impl that will be named Mock*S3Impl*.

    -   In this example, the list_objects() method is the call you want mocked. automock will automatically create an expect_*list_objects()* method for you.

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

```rust
#[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. Create the test functions in a module named `test`.

    - The conditional `#[cfg(test)]` indicates that `mockall` should build the test module if the
      `test` attribute is `true`.

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

- Each test uses `let mut mock = MockS3Impl::default();` to create a mock instance of `MockS3Impl`.

- It uses the mock's `expect_list_objects()` method (which was created automatically by `automock`) to set the expected result for when the `list_objects()` method is used elsewhere in the code.

- After the expectations are established, it uses these to test the function by calling `determine_prefix_file_size()`. The returned value is checked to confirm that it's correct, using an assertion.

6. The `determine_prefix_file_size()` function uses the Amazon S3 wrapper to get the size of the prefix file:

```
#[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)
}
```

The type S3 is used to call the wrapped SDK for Rust functions to support both S3Impl and MockS3Impl when making HTTP requests. The mock automatically generated by mockall reports any test failures when testing is enabled.

You can [view the complete code for these examples](#) on GitHub.

Simulate HTTP traffic using static replay

The aws-smithy-runtime crate includes a test utility class called [StaticReplayClient](#). This HTTP client class can be specified instead of the default HTTP client when creating an AWS service object.

When initializing the StaticReplayClient, you provide a list of HTTP request and response pairs as ReplayEvent objects. While the test is running, each HTTP request is recorded and the client returns the next HTTP response found in the next ReplayEvent in the event list as the HTTP client's response. This lets the test run using known data and without a network connection.

Using static replay

To use static replay, you don't need to use a wrapper. Instead, determine what the actual network traffic should look like for the data your test will use, and provide that traffic data to the StaticReplayClient to use each time the SDK issues a request from the AWS service client.

> ⓘ **Note**
>
> There are several ways to collect the expected network traffic, including the AWS CLI and many network traffic analyzers and packet sniffer tools.

- Create a list of `ReplayEvent` objects that specify the expected HTTP requests and the responses that should be returned for them.

- Create a `StaticReplayClient` using the HTTP transaction list created in the previous step.

- Create a configuration object for the AWS client, specifying the `StaticReplayClient` as the `Config` object's `http_client`.

- Create the AWS service client object, using the configuration created in the previous step.

- Perform the operations that you want to test, using the service object that's configured to use the `StaticReplayClient`. Each time the SDK sends an API request to AWS, the next response in the list is used.

> **ⓘ Note**
>
> The next response in the list is always returned, even if the sent request doesn't match the one in the vector of `ReplayEvent` objects.

- When all the desired requests have been made, call the `StaticReplayClient.assert_requests_match()` function to verify that the requests sent by the SDK match the ones in the list of `ReplayEvent` objects.

Example

Let's look at the tests for the same `determine_prefix_file_size()` function in the previous example, but using static replay instead of mocking.

1. In a command prompt for your project directory, add the aws-smithy-runtime crate as a dependency:

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

   This adds the crate to the [dependencies] section of your `Cargo.toml` file.

2. In your source file, include the `aws_smithy_runtime` types that you'll need.

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

3. The test begins by creating the `ReplayEvent` structures representing each of the HTTP transactions that should take place during the test. Each event contains an HTTP request

object and an HTTP response object representing the information that the AWS service would normally reply with. These events are passed into a call to `StaticReplayClient::new()`:

```rust
        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]);
```

The result is stored in `replay_client`. This represents an HTTP client that can then be used by the SDK for Rust by specifying it in the client's configuration.

4. To create the Amazon S3 client, call the client class's `from_conf()` function to create the client using a configuration object:

```rust
        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(),
        );
```

The configuration object is specified using the builder's `http_client()` method, and the credentials are specified using the `credentials_provider()` method. The credentials are created using a function called `make_s3_test_credentials()`, which returns a fake credentials structure:

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

These credentials don't need to be valid because they won't actually be sent to AWS.

5. Run the test by calling the function that needs testing. In this example, that function's name is `determine_prefix_file_size()`. Its first parameter is the Amazon S3 client object to use for its requests. Therefore, specify the client created using the `StaticReplayClient` so requests are handled by that rather than going out over the network:

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

        assert_eq!(19, size);

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

When the call to `determine_prefix_file_size()` is finished, an assert is used to confirm that the returned value matches the expected value. Then, the `StaticReplayClient` method `assert_requests_match()` function is called. This function scans the recorded HTTP requests and confirms that they all match the ones specified in the array of `ReplayEvent` objects provided when creating the replay client.

You can [view the complete code for these examples](#) on GitHub.

Waiters

Waiters are a client-side abstraction used to poll a resource until a desired state is reached, or until it is determined that the resource will not enter the desired state. This is a common task when working with services that are eventually consistent, like Amazon Simple Storage Service, or services that asynchronously create resources, like Amazon Elastic Compute Cloud. Writing logic to continuously poll the status of a resource can be cumbersome and error-prone. The goal of waiters is to move this responsibility out of customer code and into the AWS SDK for Rust, which has in-depth knowledge of the timing aspects for the AWS operation.

AWS services that provide support for waiters include a `<service>::waiters` module.

- The `<service>::client::Waiters` trait provides waiter methods for the client. The methods are implemented for the `Client` struct. All waiter methods follow a standard naming convention of `wait_until_<Condition>`

  - For Amazon S3, this trait is [`aws_sdk_s3::client::Waiters`](#).

The following example uses Amazon S3. However, the concepts are the same for any AWS service that has one or more waiters defined.

The following code example shows using a waiter function instead of writing polling logic to wait for a bucket to exist after being created.

```
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**
>
> Each wait method returns a `Result<FinalPoll<...>, WaiterError<...>>` that can
> be used to get at the final response from reaching the desired condition or an error. See
> [FinalPoll](#) and [WaiterError](#) in the Rust API documentation for details.

SDK for Rust code examples

The code examples in this topic show you how to use the AWS SDK for Rust with AWS.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Services

- [API Gateway examples using SDK for Rust](#)
- [API Gateway Management API examples using SDK for Rust](#)
- [Application Auto Scaling examples using SDK for Rust](#)
- [Aurora examples using SDK for Rust](#)
- [Auto Scaling examples using SDK for Rust](#)
- [Amazon Bedrock Runtime examples using SDK for Rust](#)
- [Amazon Cognito Identity Provider examples using SDK for Rust](#)
- [Amazon Cognito Sync examples using SDK for Rust](#)
- [Firehose examples using SDK for Rust](#)
- [Amazon DocumentDB examples using SDK for Rust](#)
- [DynamoDB examples using SDK for Rust](#)
- [Amazon EBS examples using SDK for Rust](#)
- [Amazon EC2 examples using SDK for Rust](#)
- [Amazon ECR examples using SDK for Rust](#)
- [Amazon ECS examples using SDK for Rust](#)
- [Amazon EKS examples using SDK for Rust](#)
- [AWS Glue examples using SDK for Rust](#)
- [IAM examples using SDK for Rust](#)
- [AWS IoT examples using SDK for Rust](#)
- [Kinesis examples using SDK for Rust](#)

- [AWS KMS examples using SDK for Rust](#)

- [Lambda examples using SDK for Rust](#)

- [MediaLive examples using SDK for Rust](#)

- [MediaPackage examples using SDK for Rust](#)

- [Amazon Polly examples using SDK for Rust](#)

- [QLDB examples using SDK for Rust](#)

- [Amazon RDS examples using SDK for Rust](#)

- [Amazon RDS Data Service examples using SDK for Rust](#)

- [Amazon Rekognition examples using SDK for Rust](#)

- [Route 53 examples using SDK for Rust](#)

- [Amazon S3 examples using SDK for Rust](#)

- [SageMaker examples using SDK for Rust](#)

- [Secrets Manager examples using SDK for Rust](#)

- [Amazon SES API v2 examples using SDK for Rust](#)

- [Amazon SNS examples using SDK for Rust](#)

- [Amazon SQS examples using SDK for Rust](#)

- [AWS STS examples using SDK for Rust](#)

- [Systems Manager examples using SDK for Rust](#)

- [Amazon Transcribe examples using SDK for Rust](#)

API Gateway examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with API Gateway.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

- [Scenarios](#)

Actions

GetRestApis

The following code example shows how to use `GetRestApis`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Displays the Amazon API Gateway REST APIs in the Region.

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

- For API details, see [GetRestApis](#) in *AWS SDK for Rust API reference*.

Scenarios

Create a serverless application to manage photos

The following code example shows how to create a serverless application that lets users manage photos using labels.

SDK for Rust

> Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.
>
> For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).
>
> For a deep dive into the origin of this example see the post on [AWS Community](#).
>
> **Services used in this example**
>
> - API Gateway
> - DynamoDB
> - Lambda
> - Amazon Rekognition
> - Amazon S3
> - Amazon SNS

API Gateway Management API examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with API Gateway Management API.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

PostToConnection

The following code example shows how to use PostToConnection.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```rust
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);
```

- For API details, see PostToConnection in *AWS SDK for Rust API reference*.

Application Auto Scaling examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Application Auto Scaling.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

`DescribeScalingPolicies`

The following code example shows how to use `DescribeScalingPolicies`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

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

    Ok(())
}
```

- For API details, see [DescribeScalingPolicies](#) in *AWS SDK for Rust API reference*.

Aurora examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Aurora.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Aurora

The following code examples show how to get started using Aurora.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
use aws_sdk_rds::Client;

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

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

- For API details, see DescribeDBClusters in *AWS SDK for Rust API reference*.

Topics

- Basics

- Actions

Basics

Learn the basics

The following code example shows how to:

- Create a custom Aurora DB cluster parameter group and set parameter values.

- Create a DB cluster that uses the parameter group.

- Create a DB instance that contains a database.

- Take a snapshot of the DB cluster, then clean up resources.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

A library containing the scenario-specific functions for the Aurora scenario.

```
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 {
```

```rust
                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 {
```

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

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

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

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

        if let Err(err) = describe_db_engine_versions {
```

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

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

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

        Ok(versions)
    }

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

        describe_orderable_db_instance_options_items
```

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

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

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

```rust
        }
        _ => {
            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'.
```

```rust
    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")
        );
```

```rust
        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"))
```

```rust
    }

    // 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() {
```

```rust
                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;
```

```rust
        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;
```

Tests for the library using automocks around the RDS Client wrapper.

```
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());
```

```rust
}

#[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"
```

```rust
    );
}

#[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")
```

```rust
                            .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);
```

```rust
        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());
```

```rust
    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;
}
```

```rust
#[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
```

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

A binary to run the scenario from front to end, using inquirer so that the user can make some decisions.

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

use anyhow::anyhow;
```

```rust
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)
```

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


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

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

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

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

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

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

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

    Ok(scenario)
}

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

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

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

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

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

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

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

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

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

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

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

        Ok(())
}

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

```rust
    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;
```

```rust
    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),
    }
}
```

```rust
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
        }
    }
}
```

A wrapper around the Amazon RDS service that allows automocking for tests.

```rust
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},
```

```rust
        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 }
    }
```

```rust
    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)
```

```rust
            .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
    }
```

```rust
    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()
```

```rust
                .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
    }
}
```

The Cargo.toml with dependencies used in this scenario.

```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"] }
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.

  - CreateDBCluster

  - CreateDBClusterParameterGroup

  - CreateDBClusterSnapshot

  - CreateDBInstance

  - DeleteDBCluster

  - DeleteDBClusterParameterGroup

  - DeleteDBInstance

  - DescribeDBClusterParameterGroups

  - DescribeDBClusterParameters

  - DescribeDBClusterSnapshots

  - DescribeDBClusters

  - DescribeDBEngineVersions

  - DescribeDBInstances

  - DescribeOrderableDBInstanceOptions

  - ModifyDBClusterParameterGroup

Actions

CreateDBCluster

The following code example shows how to use CreateDBCluster.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    // 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,
            ));
        }
```

```rust
        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;
```

```rust
        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;
}
```

- For API details, see [CreateDBCluster](#) in *AWS SDK for Rust API reference*.

CreateDBClusterParameterGroup

The following code example shows how to use `CreateDBClusterParameterGroup`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    // 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,
```

```rust
                ));
            }
        }
        _ => {
            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());
}
```

- For API details, see [CreateDBClusterParameterGroup](#) in *AWS SDK for Rust API reference*.

CreateDBClusterSnapshot

The following code example shows how to use `CreateDBClusterSnapshot`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

```rust
    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())
        });
```

```rust
    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);
```

```rust
    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;
```

```
    }
```

- For API details, see [CreateDBClusterSnapshot](#) in *AWS SDK for Rust API reference.*

CreateDBInstance

The following code example shows how to use `CreateDBInstance`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    // 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;
}
```

- For API details, see [CreateDBInstance](#) in *AWS SDK for Rust API reference*.

DeleteDBCluster

The following code example shows how to use `DeleteDBCluster`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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;
            }
```

```rust
                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(|_| {
```

```rust
            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;
}
```

- For API details, see [DeleteDBCluster](#) in *AWS SDK for Rust API reference*.

DeleteDBClusterParameterGroup

The following code example shows how to use `DeleteDBClusterParameterGroup`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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;
                    }
```

```rust
                }
            }
        }

        // 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
    }
```

```rust
#[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;
}
```

- For API details, see [DeleteDBClusterParameterGroup](#) in *AWS SDK for Rust API reference*.

DeleteDBInstance

The following code example shows how to use `DeleteDBInstance`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    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() {
```

```rust
                    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
```

```rust
    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;
}
```

- For API details, see DeleteDBInstance in *AWS SDK for Rust API reference*.

DescribeDBClusterParameters

The following code example shows how to use DescribeDBClusterParameters.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

```
    // 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.
```

```rust
    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() {
```

```rust
    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");
}
```

- For API details, see [DescribeDBClusterParameters](#) in *AWS SDK for Rust API reference*.

DescribeDBClusters

The following code example shows how to use `DescribeDBClusters`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

```rust
    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");
```

```rust
            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;
}
```

- For API details, see [DescribeDBClusters](#) in *AWS SDK for Rust API reference.*

DescribeDBEngineVersions

The following code example shows how to use `DescribeDBEngineVersions`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };

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

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
```

```
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }

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

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

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

```
                    .db_engine_versions(
                        DbEngineVersion::builder()
                            .db_parameter_group_family("f1")
                            .engine_version("f1b")
                            .build(),
                    )
                    .db_engine_versions(
                        DbEngineVersion::builder()
                            .db_parameter_group_family("f2")
                            .engine_version("f2a")
                            .build(),
                    )
                    .db_engine_versions(DbEngineVersion::builder().build())
                    .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

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

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

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

```
        });

    let scenario = AuroraScenario::new(mock_rds);

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

- For API details, see [DescribeDBEngineVersions](#) in *AWS SDK for Rust API reference.*

DescribeDBInstances

The following code example shows how to use `DescribeDBInstances`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
    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.
```

```rust
        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(
```

```rust
                    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;
}
```

- For API details, see [DescribeDBInstances](#) in *AWS SDK for Rust API reference.*

DescribeOrderableDBInstanceOptions

The following code example shows how to use DescribeOrderableDBInstanceOptions.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
    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
```

```rust
                .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()
```

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

- For API details, see [DescribeOrderableDBInstanceOptions](#) in *AWS SDK for Rust API reference*.

ModifyDBClusterParameterGroup

The following code example shows how to use `ModifyDBClusterParameterGroup`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    // 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(
```

```rust
        &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");
}
```

- For API details, see [ModifyDBClusterParameterGroup](#) in *AWS SDK for Rust API reference*.

Auto Scaling examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Auto Scaling.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Auto Scaling

The following code examples show how to get started using Auto Scaling.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DescribeAutoScalingGroups](#) in *AWS SDK for Rust API reference.*

Topics

- [Basics](#)
- [Actions](#)

Basics

Learn the basics

The following code example shows how to:

- Create an Amazon EC2 Auto Scaling group with a launch template and Availability Zones, and get information about running instances.
- Enable Amazon CloudWatch metrics collection.
- Update the group's desired capacity and wait for an instance to start.
- Terminate an instance in the group.
- List scaling activities that occur in response to user requests and capacity changes.
- Get statistics for CloudWatch metrics, then clean up resources.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
[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> {
```

```rust
    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);
        }
```

```rust
    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);
```

```rust
    }
    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";
```

```rust
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 {
```

```rust
            f.write_fmt(format_args!(
                "\tLaunch Template ID: {}\n",
                self.launch_template_arn
            ))?;
            f.write_fmt(format_args!(
                "\tScaling Group Name: {}\n",
                self.auto_scaling_group_name
            ))?;

            Ok(())
        }
    }

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

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

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

```rust
                    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}")
    }
}
```

```rust
#[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>> {
```

```rust
        let ec2 = aws_sdk_ec2::Client::new(sdk_config);
        let autoscaling = aws_sdk_autoscaling::Client::new(sdk_config);

        let auto_scaling_group_name = String::from(AUTOSCALING_GROUP_NAME);

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

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

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

        let launch_template_arn = match create_launch_template.launch_template {
            Some(launch_template) =>
launch_template.launch_template_id.unwrap_or_default(),
            None => {
                // Try to delete the launch template
                let _ = ec2
```

```
                        .delete_launch_template()
                        .launch_template_name(LAUNCH_TEMPLATE_NAME)
                        .send()
                        .await;
                    return Err(vec![ScenarioError::with("Failed to load launch
template")]);
                }
        };

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

            if let Err(err) = autoscaling
                .delete_auto_scaling_group()
                .auto_scaling_group_name(auto_scaling_group_name.as_str())
                .send()
                .await
            {
                errs.push(ScenarioError::new(
                    "Failed to clean up autoscaling group",
                    &err,
```

```
            ));
        }

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

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

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

    match enable_metrics_collection {
        Ok(_) => Ok(scenario),
        Err(err) => {
            scenario.clean_scenario().await?;
            Err(vec![ScenarioError::new(
```

```
                    "Failed to enable metrics collections for group",
                    &err,
                )])
            }
        }
    }

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

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

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

        if early_exit.is_err() {
            early_exit
```

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

```rust
    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()
```

```rust
                .collect::<Result<Vec<_>, _>>()
                .await
                .map_err(|e| {
                    anyhow!(
                        "There was an error retrieving scaling activities: {}",
                        DisplayErrorContext(&e)
                    )
                });

        AutoScalingScenarioDescription {
            group,
            instances,
            activities,
        }
    }

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

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

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

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

```rust
                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;
```

```rust
            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.
```

```rust
        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)
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.

  - [CreateAutoScalingGroup](#)

  - [DeleteAutoScalingGroup](#)

  - [DescribeAutoScalingGroups](#)

  - [DescribeAutoScalingInstances](#)

  - [DescribeScalingActivities](#)

- DisableMetricsCollection

- EnableMetricsCollection

- SetDesiredCapacity

- TerminateInstanceInAutoScalingGroup

- UpdateAutoScalingGroup

Actions

CreateAutoScalingGroup

The following code example shows how to use `CreateAutoScalingGroup`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for Rust API reference.*

DeleteAutoScalingGroup

The following code example shows how to use DeleteAutoScalingGroup.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for Rust API reference*.

DescribeAutoScalingGroups

The following code example shows how to use DescribeAutoScalingGroups.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see [DescribeAutoScalingGroups](#) in *AWS SDK for Rust API reference.*

DescribeAutoScalingInstances

The following code example shows how to use DescribeAutoScalingInstances.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
    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))
    }
```

- For API details, see [DescribeAutoScalingInstances](#) in *AWS SDK for Rust API reference*.

DescribeScalingActivities

The following code example shows how to use DescribeScalingActivities.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

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

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

    // 10. DescribeScalingActivities: list the scaling activities that have
occurred for the group so far.
    //    Bonus: use CloudWatch API to get and show some metrics collected for
the group.
```

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

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

- For API details, see DescribeScalingActivities in *AWS SDK for Rust API reference.*

DisableMetricsCollection

The following code example shows how to use DisableMetricsCollection.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        // 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;
```

- For API details, see DisableMetricsCollection in *AWS SDK for Rust API reference.*

EnableMetricsCollection

The following code example shows how to use `EnableMetricsCollection`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

```
        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;
```

- For API details, see EnableMetricsCollection in *AWS SDK for Rust API reference.*

SetDesiredCapacity

The following code example shows how to use SetDesiredCapacity.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [SetDesiredCapacity](#) in *AWS SDK for Rust API reference*.

TerminateInstanceInAutoScalingGroup

The following code example shows how to use TerminateInstanceInAutoScalingGroup.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

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

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

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

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

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

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

- For API details, see [TerminateInstanceInAutoScalingGroup](#) in *AWS SDK for Rust API reference*.

UpdateAutoScalingGroup

The following code example shows how to use UpdateAutoScalingGroup.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [UpdateAutoScalingGroup](#) in *AWS SDK for Rust API reference*.

Amazon Bedrock Runtime examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon Bedrock Runtime.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Anthropic Claude](#)

Anthropic Claude

Converse

The following code example shows how to send a text message to Anthropic Claude, using Bedrock's Converse API.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Anthropic Claude, using Bedrock's Converse API.

```rust
#[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)
}
```

Use statements, Error utility, and constants.

```
use aws_config::BehaviorVersion;
use aws_sdk_bedrockruntime::{
    operation::converse::{ConverseError, ConverseOutput},
    types::{ContentBlock, ConversationRole, Message},
    Client,
};

// Set the model ID, e.g., Claude 3 Haiku.
const MODEL_ID: &str = "anthropic.claude-3-haiku-20240307-v1:0";
const CLAUDE_REGION: &str = "us-east-1";

// Start a conversation with the user message.
```

```
const USER_MESSAGE: &str = "Describe the purpose of a 'hello world' program in one
 line.";

#[derive(Debug)]
struct BedrockConverseError(String);
impl std::fmt::Display for BedrockConverseError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "Can't invoke '{}'. Reason: {}", MODEL_ID, self.0)
    }
}
impl std::error::Error for BedrockConverseError {}
impl From<&str> for BedrockConverseError {
    fn from(value: &str) -> Self {
        BedrockConverseError(value.to_string())
    }
}
impl From<&ConverseError> for BedrockConverseError {
    fn from(value: &ConverseError) -> Self {
        BedrockConverseError::from(match value {
            ConverseError::ModelTimeoutException(_) => "Model took too long",
            ConverseError::ModelNotReadyException(_) => "Model is not ready",
            _ => "Unknown",
        })
    }
}
```

- For API details, see Converse in *AWS SDK for Rust API reference*.

ConverseStream

The following code example shows how to send a text message to Anthropic Claude, using
Bedrock's Converse API and process the response stream in real-time.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

Send a text message to Anthropic Claude and stream reply tokens, using Bedrock's
ConverseStream API.

```rust
#[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(),
        })
}
```

Use statements, Error utility, and constants.

```
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";
```

```rust
// 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(),
            ),
        }
    }
}
```

- For API details, see [ConverseStream](#) in *AWS SDK for Rust API reference*.

Scenario: Tool use with the Converse API

The following code example shows how to build a typical interaction between an application, a generative AI model, and connected tools or APIs to mediate interactions between the AI and the outside world. It uses the example of connecting an external weather API to the AI model so it can provide real-time weather information based on user input.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

The primary scenario and logic for the demo. This orchestrates the conversation between the user, the Amazon Bedrock Converse API, and a weather tool.

```
#[derive(Debug)]
#[allow(dead_code)]
struct InvokeToolResult(String, ToolResultBlock);
struct ToolUseScenario {
    client: Client,
    conversation: Vec<Message>,
    system_prompt: SystemContentBlock,
    tool_config: ToolConfiguration,
}
```

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

```rust
    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();
}
```

The weather tool used by the demo. This script defines the tool specification and implements
the logic to retrieve weather data using from the Open-Meteo API.

```
const ENDPOINT: &str = "https://api.open-meteo.com/v1/forecast";
async fn fetch_weather_data(
```

```
        tool_use: &ToolUseBlock,
) -> Result<ToolResultBlock, ToolUseScenarioError> {
    let input = tool_use.input();
    let latitude = input
        .as_object()
        .unwrap()
        .get("latitude")
        .unwrap()
        .as_string()
        .unwrap();
    let longitude = input
        .as_object()
        .unwrap()
        .get("longitude")
        .unwrap()
        .as_string()
        .unwrap();
    let params = [
        ("latitude", latitude),
        ("longitude", longitude),
        ("current_weather", "true"),
    ];

    debug!("Calling {ENDPOINT} with {params:?}");

    let response = reqwest::Client::new()
        .get(ENDPOINT)
        .query(&params)
        .send()
        .await
        .map_err(|e| ToolUseScenarioError(format!("Error requesting weather:
{e:?}")))?
        .error_for_status()
        .map_err(|e| ToolUseScenarioError(format!("Failed to request weather:
{e:?}")))?;

    debug!("Response: {response:?}");

    let bytes = response
        .bytes()
        .await
        .map_err(|e| ToolUseScenarioError(format!("Error reading response:
{e:?}")))?;
```

```
    let result = String::from_utf8(bytes.to_vec())
        .map_err(|_| ToolUseScenarioError("Response was not utf8".into()))?;

    Ok(ToolResultBlock::builder()
        .tool_use_id(tool_use.tool_use_id())
        .content(ToolResultContentBlock::Text(result))
        .build()?)
}
```

Utilities to print the Message Content Blocks

```
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:?})"
        )))
    }
}
```

Use statements, Error utility, and constants.

```
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;
```

```rust
use tracing::debug;

/// This demo illustrates a tool use scenario using Amazon Bedrock's Converse API
 and a weather tool.
/// The script interacts with a foundation model on Amazon Bedrock to provide
 weather information based on user
/// input. It uses the Open-Meteo API (https://open-meteo.com) to retrieve current
 weather data for a given location.

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

- For API details, see [Converse](#) in *AWS SDK for Rust API reference*.

Amazon Cognito Identity Provider examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon Cognito Identity Provider.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

ListUserPools

The following code example shows how to use `ListUserPools`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [ListUserPools](#) in *AWS SDK for Rust API reference*.

Amazon Cognito Sync examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon Cognito Sync.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

ListIdentityPoolUsage

The following code example shows how to use `ListIdentityPoolUsage`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [ListIdentityPoolUsage](#) in *AWS SDK for Rust API reference.*

Firehose examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Firehose.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

PutRecordBatch

The following code example shows how to use `PutRecordBatch`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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
}
```

- For API details, see [PutRecordBatch](#) in *AWS SDK for Rust API reference*.

Amazon DocumentDB examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon DocumentDB.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Serverless examples](#)

Serverless examples

Invoke a Lambda function from a Amazon DocumentDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DocumentDB change stream. The function retrieves the DocumentDB payload and logs the record contents.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming a Amazon DocumentDB event with Lambda using Rust.

```rust
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(())

}
```

DynamoDB examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with DynamoDB.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)
- [Scenarios](#)
- [Serverless examples](#)

Actions

CreateTable

The following code example shows how to use `CreateTable`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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))
        }
    }
}
```

- For API details, see [CreateTable](#) in *AWS SDK for Rust API reference*.

DeleteItem

The following code example shows how to use DeleteItem.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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)),
    }
}
```

- For API details, see [DeleteItem](#) in *AWS SDK for Rust API reference*.

DeleteTable

The following code example shows how to use DeleteTable.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DeleteTable](#) in *AWS SDK for Rust API reference*.

ListTables

The following code example shows how to use `ListTables`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect::<Result<Vec<_>, _>>().await?;
```

```
    println!("Tables:");

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

    println!("Found {} tables", table_names.len());
    Ok(table_names)
}
```

Determine whether table exists.

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

- For API details, see [ListTables](#) in *AWS SDK for Rust API reference*.

PutItem

The following code example shows how to use `PutItem`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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,
    })
}
```

- For API details, see PutItem in *AWS SDK for Rust API reference.*

Query

The following code example shows how to use `Query`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Find the movies made in the specified year.

```rust
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![])
    }
}
```

- For API details, see [Query](#) in *AWS SDK for Rust API reference*.

Scan

The following code example shows how to use `Scan`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

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

- For API details, see [Scan](#) in *AWS SDK for Rust API reference*.

Scenarios

Connect to a local instance

The following code example shows how to override an endpoint URL to connect to a local
development deployment of DynamoDB and an AWS SDK.

For more information, see [DynamoDB Local](#).

SDK for Rust

> 🛈 **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

Create a serverless application to manage photos

The following code example shows how to create a serverless application that lets users manage photos using labels.

SDK for Rust

> Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.
>
> For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).
>
> For a deep dive into the origin of this example see the post on [AWS Community](#).
>
> **Services used in this example**
>
> - API Gateway
> - DynamoDB
> - Lambda
> - Amazon Rekognition
> - Amazon S3
> - Amazon SNS

Query a table using PartiQL

The following code example shows how to:

- Get an item by running a SELECT statement.
- Add an item by running an INSERT statement.
- Update an item by running an UPDATE statement.
- Delete an item by running a DELETE statement.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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),
```

```rust
    }
}

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

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
```

```
                    println!("Found a matching entry in the table:");
                    println!("{:?}", resp.items.unwrap_or_default().pop());
                    true
                } else {
                    println!("Did not find a match.");
                    false
                }
            }
            Err(e) => {
                println!("Got an error querying table:");
                println!("{}", e);
                process::exit(1);
            }
        }
    }
}

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

    println!("Deleted item.");

    Ok(())
}

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

    Ok(())
}
```

- For API details, see [ExecuteStatement](#) in *AWS SDK for Rust API reference*.

Save EXIF and other image information

The following code example shows how to:

- Get EXIF information from a a JPG, JPEG, or PNG file.

- Upload the image file to an Amazon S3 bucket.

- Use Amazon Rekognition to identify the three top attributes (labels) in the file.

- Add the EXIF and label information to an Amazon DynamoDB table in the Region.

SDK for Rust

Get EXIF information from a JPG, JPEG, or PNG file, upload the image file to an Amazon S3 bucket, use Amazon Rekognition to identify the three top attributes (*labels* in Amazon Rekognition) in the file, and add the EXIF and label information to a Amazon DynamoDB table in the Region.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- DynamoDB

- Amazon Rekognition

- Amazon S3

Serverless examples

Invoke a Lambda function from a DynamoDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DynamoDB stream. The function retrieves the DynamoDB payload and logs the record contents.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a DynamoDB event with Lambda using Rust.

```rust
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(())

}
```

Reporting batch item failures for Lambda functions with a DynamoDB trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a DynamoDB stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Reporting DynamoDB batch item failures with Lambda using Rust.

```
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;
```

```rust
    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);


    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed item
 onwards. */
            return Ok(response);
        }
    }
```

```
    tracing::info!("Successfully processed {} record(s)", records.len());


    Ok(response)
}


#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Amazon EBS examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon EBS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

CompleteSnapshot

The following code example shows how to use CompleteSnapshot.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [CompleteSnapshot](#) in *AWS SDK for Rust API reference.*

PutSnapshotBlock

The following code example shows how to use PutSnapshotBlock.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [PutSnapshotBlock](#) in *AWS SDK for Rust API reference*.

StartSnapshot

The following code example shows how to use `StartSnapshot`.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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())
}
```

- For API details, see StartSnapshot in *AWS SDK for Rust API reference*.

Amazon EC2 examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon EC2.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Amazon EC2

The following code examples show how to get started using Amazon EC2.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see [DescribeSecurityGroups](#) in *AWS SDK for Rust API reference*.

Topics

- [Basics](#)
- [Actions](#)

Basics

Learn the basics

The following code example shows how to:

- Create a key pair and security group.
- Select an Amazon Machine Image (AMI) and compatible instance type, then create an instance.

- Stop and restart the instance.

- Associate an Elastic IP address with your instance.

- Connect to your instance with SSH, then clean up resources.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

The EC2InstanceScenario implementation contains logic to run the example as a whole.

```rust
//! 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},
```

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

```rust
    /// 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
```

```rust
            {
                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?;
```

```rust
        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!
("-------------------------------------------------------------------");
}
```

The EC2Impl struct serves as a an automock point for testing, and its functions wrap the EC2 SDK calls.

```rust
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_pair_id: &str) -> Result<(), EC2Error> {
        let key_pair_id: String = key_pair_id.into();
        tracing::info!("Deleting key pair {key_pair_id}");
        self.client
            .delete_key_pair()
            .key_pair_id(key_pair_id)
            .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
```

```rust
                .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())
```

```rust
    }

    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)
    }
}
```

The SSM struct serves as a an automock point for testing, and its functions wraps SSM SDK calls.

```rust
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)
    }
}
```

The scenario uses several "Manager"-style structs to handle access to resources that are created and deleted throughout the scenario.

```
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 {
```

```rust
        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| e.add_message(format!("Couldn't create key {key_name}")))?;

        let path = self.key_file_dir.join(format!("{key_name}.pem"));
```

```rust
        util.write_secure(&key_name, &path, material)?;

        self.key_file_path = Some(path);
        self.key_pair = key_pair.clone();

        Ok(key_pair)
    }

    pub async fn delete(self, ec2: &EC2, util: &Util) -> Result<(), EC2Error> {
        if let Some(key_pair_id) = self.key_pair.key_pair_id() {
            ec2.delete_key_pair(key_pair_id).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,
```

```rust
/// 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(
```

```rust
                    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."),
```

```
        }
    }
}
```

The main entry point for the scenario.

```
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;
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.

  - [AllocateAddress](#)

  - [AssociateAddress](#)

  - [AuthorizeSecurityGroupIngress](#)

  - [CreateKeyPair](#)

  - [CreateSecurityGroup](#)

  - [DeleteKeyPair](#)

  - [DeleteSecurityGroup](#)

  - [DescribeImages](#)

  - [DescribeInstanceTypes](#)

- DescribeInstances

- DescribeKeyPairs

- DescribeSecurityGroups

- DisassociateAddress

- ReleaseAddress

- RunInstances

- StartInstances

- StopInstances

- TerminateInstances

- UnmonitorInstances

Actions

AllocateAddress

The following code example shows how to use `AllocateAddress`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see AllocateAddress in *AWS SDK for Rust API reference*.

AssociateAddress

The following code example shows how to use `AssociateAddress`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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)
}
```

- For API details, see AssociateAddress in *AWS SDK for Rust API reference*.

AuthorizeSecurityGroupIngress

The following code example shows how to use `AuthorizeSecurityGroupIngress`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see [AuthorizeSecurityGroupIngress](#) in *AWS SDK for Rust API reference*.

CreateKeyPair

The following code example shows how to use `CreateKeyPair`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Rust implementation that calls the EC2 Client's create_key_pair and extracts the returned material.

```rust
    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))
    }
```

A function that calls the create_key impl and securely saves the PEM private key.

```rust
    /// 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| e.add_message(format!("Couldn't create key {key_name}"))))?;

        let path = self.key_file_dir.join(format!("{key_name}.pem"));

        util.write_secure(&key_name, &path, material)?;

        self.key_file_path = Some(path);
        self.key_pair = key_pair.clone();

        Ok(key_pair)
    }
```

- For API details, see [CreateKeyPair](#) in *AWS SDK for Rust API reference.*

CreateSecurityGroup

The following code example shows how to use `CreateSecurityGroup`.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
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)
    }
```

- For API details, see [CreateSecurityGroup](#) in *AWS SDK for Rust API reference*.

CreateTags

The following code example shows how to use `CreateTags`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

This example applies the Name tag After creating an instance.

```
    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())
}
```

- For API details, see CreateTags in *AWS SDK for Rust API reference*.

DeleteKeyPair

The following code example shows how to use `DeleteKeyPair`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Wrapper around delete_key that also removes the backing private PEM key.

```
pub async fn delete(self, ec2: &EC2, util: &Util) -> Result<(), EC2Error> {
    if let Some(key_pair_id) = self.key_pair.key_pair_id() {
        ec2.delete_key_pair(key_pair_id).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_pair_id: &str) -> Result<(), EC2Error> {
        let key_pair_id: String = key_pair_id.into();
        tracing::info!("Deleting key pair {key_pair_id}");
        self.client
            .delete_key_pair()
            .key_pair_id(key_pair_id)
            .send()
            .await?;
        Ok(())
    }
```

- For API details, see [DeleteKeyPair](#) in *AWS SDK for Rust API reference.*

DeleteSecurityGroup

The following code example shows how to use `DeleteSecurityGroup`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    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(())
    }
```

- For API details, see [DeleteSecurityGroup](#) in *AWS SDK for Rust API reference.*

DeleteSnapshot

The following code example shows how to use `DeleteSnapshot`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

    println!("Deleted");

    Ok(())
}
```

- For API details, see [DeleteSnapshot](#) in *AWS SDK for Rust API reference*.

DescribeImages

The following code example shows how to use `DescribeImages`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    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)
        }
    }
```

Using the list_images function with SSM to limit based on your environment. For more details on SSM, see 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)
    }
```

- For API details, see DescribeImages in *AWS SDK for Rust API reference*.

DescribeInstanceStatus

The following code example shows how to use `DescribeInstanceStatus`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DescribeInstanceStatus](#) in *AWS SDK for Rust API reference*.

DescribeInstanceTypes

The following code example shows how to use `DescribeInstanceTypes`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DescribeInstanceTypes](#) in *AWS SDK for Rust API reference*.

DescribeInstances

The following code example shows how to use `DescribeInstances`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Retrieve details for an EC2 Instance.

```
    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())
    }
```

After creating an EC2 instance, retrieve and store its details.

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

- For API details, see DescribeInstances in *AWS SDK for Rust API reference*.

DescribeKeyPairs

The following code example shows how to use `DescribeKeyPairs`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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())
}
```

- For API details, see [DescribeKeyPairs](#) in *AWS SDK for Rust API reference.*

DescribeRegions

The following code example shows how to use `DescribeRegions`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see DescribeRegions in *AWS SDK for Rust API reference*.

DescribeSecurityGroups

The following code example shows how to use `DescribeSecurityGroups`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

```
}
```

- For API details, see [DescribeSecurityGroups](#) in *AWS SDK for Rust API reference*.

DescribeSnapshots

The following code example shows how to use `DescribeSnapshots`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Shows the state of a snapshot.

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

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

- For API details, see DescribeSnapshots in *AWS SDK for Rust API reference.*

DisassociateAddress

The following code example shows how to use DisassociateAddress.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    pub async fn disassociate_ip_address(&self, association_id: &str) -> Result<(),
EC2Error> {
        self.client
            .disassociate_address()
            .association_id(association_id)
```

```
            .send()
            .await?;
        Ok(())
    }
```

- For API details, see [DisassociateAddress](#) in *AWS SDK for Rust API reference*.

RebootInstances

The following code example shows how to use `RebootInstances`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

```rust
pub async fn reboot_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Rebooting instance {instance_id}");

    self.client
        .reboot_instances()
        .instance_ids(instance_id)
        .send()
        .await?;
```

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

Waiters for instance to be in the stopped and ready states, using the Waiters API. Using the
Waiters API requires `use aws_sdk_ec2::client::Waiters` in the rust file.

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

- For API details, see [RebootInstances](#) in *AWS SDK for Rust API reference*.

ReleaseAddress

The following code example shows how to use `ReleaseAddress`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [ReleaseAddress](#) in *AWS SDK for Rust API reference*.

RunInstances

The following code example shows how to use `RunInstances`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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())
}
```

- For API details, see [RunInstances](#) in *AWS SDK for Rust API reference*.

StartInstances

The following code example shows how to use `StartInstances`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Start an EC2 Instance by instance ID.

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

Wait for an instance to be in the ready and status ok states, using the Waiters API. Using the Waiters API requires `use aws_sdk_ec2::client::Waiters` in the rust file.

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

- For API details, see StartInstances in *AWS SDK for Rust API reference.*

StopInstances

The following code example shows how to use `StopInstances`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

Wait for an instance to be in the stopped state, using the Waiters API. Using the Waiters API requires `use aws_sdk_ec2::client::Waiters` in the rust file.

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

- For API details, see StopInstances in *AWS SDK for Rust API reference.*

TerminateInstances

The following code example shows how to use `TerminateInstances`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

```
    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(())
    }
```

Wait for an instance to be in the terminted state, using the Waiters API. Using the Waiters API
requires `use aws_sdk_ec2::client::Waiters` in the rust file.

```
    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(())
    }
```

- For API details, see [TerminateInstances](#) in *AWS SDK for Rust API reference*.

Amazon ECR examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon ECR.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

DescribeRepositories

The following code example shows how to use `DescribeRepositories`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DescribeRepositories](#) in *AWS SDK for Rust API reference.*

ListImages

The following code example shows how to use `ListImages`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [ListImages](#) in *AWS SDK for Rust API reference*.

Amazon ECS examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon ECS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

CreateCluster

The following code example shows how to use `CreateCluster`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [CreateCluster](#) in *AWS SDK for Rust API reference*.

DeleteCluster

The following code example shows how to use `DeleteCluster`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [DeleteCluster](#) in *AWS SDK for Rust API reference*.

DescribeClusters

The following code example shows how to use `DescribeClusters`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
```

```
}
```

- For API details, see DescribeClusters in *AWS SDK for Rust API reference*.

Amazon EKS examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon EKS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

CreateCluster

The following code example shows how to use `CreateCluster`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see CreateCluster in *AWS SDK for Rust API reference*.

DeleteCluster

The following code example shows how to use DeleteCluster.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see DeleteCluster in *AWS SDK for Rust API reference*.

AWS Glue examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with AWS Glue.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello AWS Glue

The following code examples show how to get started using AWS Glue.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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)),
    }
}
```

- For API details, see ListJobs in *AWS SDK for Rust API reference*.

Topics

- Basics
- Actions

Basics

Learn the basics

The following code example shows how to:

- Create a crawler that crawls a public Amazon S3 bucket and generates a database of CSV-formatted metadata.
- List information about databases and tables in your AWS Glue Data Catalog.
- Create a job to extract CSV data from the S3 bucket, transform the data, and load JSON-formatted output into another S3 bucket.
- List information about job runs, view transformed data, and clean up resources.

For more information, see Tutorial: Getting started with AWS Glue Studio.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create and run a crawler that crawls a public Amazon Simple Storage Service (Amazon S3) bucket and generates a metadata database that describes the CSV-formatted data it finds.

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

List information about databases and tables in your 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();
```

Create and run a job that extracts CSV data from the source Amazon S3 bucket, transforms it by removing and renaming fields, and loads JSON-formatted output into another Amazon S3 bucket.

```
        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();
```

Delete all resources created by the demo.

```
        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)?;
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.

  - [CreateCrawler](#)

  - [CreateJob](#)

  - [DeleteCrawler](#)

  - [DeleteDatabase](#)

  - [DeleteJob](#)

  - [DeleteTable](#)

  - [GetCrawler](#)

  - [GetDatabase](#)

  - [GetDatabases](#)

  - [GetJob](#)

  - [GetJobRun](#)

  - [GetJobRuns](#)

  - [GetTables](#)

  - [ListJobs](#)

  - [StartCrawler](#)

  - [StartJobRun](#)

Actions

CreateCrawler

The following code example shows how to use `CreateCrawler`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

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

- For API details, see [CreateCrawler](#) in *AWS SDK for Rust API reference*.

CreateJob

The following code example shows how to use `CreateJob`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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())
})?;
```

- For API details, see [CreateJob](#) in *AWS SDK for Rust API reference*.

DeleteCrawler

The following code example shows how to use `DeleteCrawler`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DeleteCrawler](#) in *AWS SDK for Rust API reference.*

DeleteDatabase

The following code example shows how to use `DeleteDatabase`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DeleteDatabase](#) in *AWS SDK for Rust API reference.*

DeleteJob

The following code example shows how to use `DeleteJob`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DeleteJob](#) in *AWS SDK for Rust API reference*.

DeleteTable

The following code example shows how to use `DeleteTable`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
for t in &self.tables {
    glue.delete_table()
        .name(t.name())
        .database_name(self.database())
        .send()
        .await
```

```
              .map_err(GlueMvpError::from_glue_sdk)?;
      }
```

- For API details, see DeleteTable in *AWS SDK for Rust API reference*.

GetCrawler

The following code example shows how to use GetCrawler.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see GetCrawler in *AWS SDK for Rust API reference*.

GetDatabase

The following code example shows how to use GetDatabase.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see [GetDatabase](#) in *AWS SDK for Rust API reference*.

GetJobRun

The following code example shows how to use GetJobRun.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
        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();
        }
```

- For API details, see [GetJobRun](#) in *AWS SDK for Rust API reference.*

GetTables

The following code example shows how to use GetTables.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

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

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

- For API details, see GetTables in *AWS SDK for Rust API reference*.

ListJobs

The following code example shows how to use `ListJobs`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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)),
    }
}
```

- For API details, see ListJobs in *AWS SDK for Rust API reference*.

StartCrawler

The following code example shows how to use `StartCrawler`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see [StartCrawler](#) in *AWS SDK for Rust API reference*.

StartJobRun

The following code example shows how to use `StartJobRun`.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
        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();
```

- For API details, see StartJobRun in *AWS SDK for Rust API reference*.

IAM examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with IAM.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello IAM

The following code examples show how to get started using IAM.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

From src/bin/hello.rs.

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

From src/iam-service-lib.rs.

```rust
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)
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for Rust API reference.*

Topics

- [Basics](#)
- [Actions](#)

Basics

Learn the basics

The following code example shows how to create a user and assume a role.

> ⚠️ **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

- Create a user with no permissions.
- Create a role that grants permission to list Amazon S3 buckets for the account.
- Add a policy to let the user assume the role.
- Assume the role and list S3 buckets using temporary credentials, then clean up resources.

SDK for Rust

> ℹ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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);
```

```rust
    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);
```

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

- For API details, see the following topics in *AWS SDK for Rust API reference*.

  - AttachRolePolicy

  - CreateAccessKey

  - CreatePolicy

  - CreateRole

  - CreateUser

  - DeleteAccessKey

  - DeletePolicy

  - DeleteRole

  - DeleteUser

  - DeleteUserPolicy

  - DetachRolePolicy

  - PutUserPolicy

Actions

AttachRolePolicy

The following code example shows how to use `AttachRolePolicy`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
}
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for Rust API reference.*

AttachUserPolicy

The following code example shows how to use `AttachUserPolicy`.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [AttachUserPolicy](#) in *AWS SDK for Rust API reference*.

CreateAccessKey

The following code example shows how to use `CreateAccessKey`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```rust
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())
}
```

- For API details, see CreateAccessKey in *AWS SDK for Rust API reference*.

CreatePolicy

The following code example shows how to use CreatePolicy.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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())
}
```

- For API details, see CreatePolicy in *AWS SDK for Rust API reference*.

CreateRole

The following code example shows how to use CreateRole.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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())
}
```

- For API details, see [CreateRole](#) in *AWS SDK for Rust API reference*.

CreateServiceLinkedRole

The following code example shows how to use `CreateServiceLinkedRole`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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)
}
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for Rust API reference*.

CreateUser

The following code example shows how to use CreateUser.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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())
}
```

- For API details, see CreateUser in *AWS SDK for Rust API reference.*

DeleteAccessKey

The following code example shows how to use DeleteAccessKey.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for Rust API reference*.

DeletePolicy

The following code example shows how to use DeletePolicy.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DeletePolicy](#) in *AWS SDK for Rust API reference*.

DeleteRole

The following code example shows how to use DeleteRole.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DeleteRole](#) in *AWS SDK for Rust API reference*.

DeleteServiceLinkedRole

The following code example shows how to use DeleteServiceLinkedRole.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see DeleteServiceLinkedRole in *AWS SDK for Rust API reference.*

DeleteUser

The following code example shows how to use `DeleteUser`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
}
```

- For API details, see [DeleteUser](#) in *AWS SDK for Rust API reference*.

DeleteUserPolicy

The following code example shows how to use `DeleteUserPolicy`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
    }
```

- For API details, see DeleteUserPolicy in *AWS SDK for Rust API reference.*

DetachRolePolicy

The following code example shows how to use DetachRolePolicy.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see DetachRolePolicy in *AWS SDK for Rust API reference.*

DetachUserPolicy

The following code example shows how to use DetachUserPolicy.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DetachUserPolicy](#) in *AWS SDK for Rust API reference*.

GetAccountPasswordPolicy

The following code example shows how to use GetAccountPasswordPolicy.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
pub async fn get_account_password_policy(
```

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

    Ok(response)
}
```

- For API details, see [GetAccountPasswordPolicy](#) in *AWS SDK for Rust API reference*.

GetRole

The following code example shows how to use GetRole.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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)
}
```

- For API details, see [GetRole](#) in *AWS SDK for Rust API reference*.

ListAttachedRolePolicies

The following code example shows how to use ListAttachedRolePolicies.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

```rust
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)
}
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for Rust API reference*.

ListGroups

The following code example shows how to use ListGroups.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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)
}
```

- For API details, see [ListGroups](#) in *AWS SDK for Rust API reference*.

ListPolicies

The following code example shows how to use `ListPolicies`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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)
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for Rust API reference.*

ListRolePolicies

The following code example shows how to use `ListRolePolicies`.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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)
}
```

- For API details, see ListRolePolicies in *AWS SDK for Rust API reference*.

ListRoles

The following code example shows how to use ListRoles.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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)
    }
```

- For API details, see [ListRoles](#) in *AWS SDK for Rust API reference*.

ListSAMLProviders

The following code example shows how to use `ListSAMLProviders`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn list_saml_providers(
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}
```

- For API details, see [ListSAMLProviders](#) in *AWS SDK for Rust API reference*.

ListUsers

The following code example shows how to use `ListUsers`.

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```rust
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)
}
```

- For API details, see ListUsers in *AWS SDK for Rust API reference*.

AWS IoT examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with AWS IoT.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

DescribeEndpoint

The following code example shows how to use `DescribeEndpoint`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [DescribeEndpoint](#) in *AWS SDK for Rust API reference*.

ListThings

The following code example shows how to use `ListThings`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

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

- For API details, see ListThings in *AWS SDK for Rust API reference*.

Kinesis examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios
by using the AWS SDK for Rust with Kinesis.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)
- [Serverless examples](#)

Actions

CreateStream

The following code example shows how to use `CreateStream`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [CreateStream](#) in *AWS SDK for Rust API reference*.

DeleteStream

The following code example shows how to use `DeleteStream`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- For API details, see [DeleteStream](#) in *AWS SDK for Rust API reference.*

DescribeStream

The following code example shows how to use `DescribeStream`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see DescribeStream in *AWS SDK for Rust API reference.*

ListStreams

The following code example shows how to use `ListStreams`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see ListStreams in *AWS SDK for Rust API reference*.

PutRecord

The following code example shows how to use `PutRecord`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see PutRecord in *AWS SDK for Rust API reference*.

Serverless examples

Invoke a Lambda function from a Kinesis trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a Kinesis stream. The function retrieves the Kinesis payload, decodes from Base64, and logs the record contents.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an Kinesis event with Lambda using Rust.

```rust
// 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
}
```

Reporting batch item failures for Lambda functions with a Kinesis trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a Kinesis stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Reporting Kinesis batch item failures with Lambda using Rust.

```rust
// 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 examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with AWS KMS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

CreateKey

The following code example shows how to use `CreateKey`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [CreateKey](#) in *AWS SDK for Rust API reference*.

Decrypt

The following code example shows how to use `Decrypt`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [Decrypt](#) in *AWS SDK for Rust API reference*.

Encrypt

The following code example shows how to use `Encrypt`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [Encrypt](#) in *AWS SDK for Rust API reference*.

GenerateDataKey

The following code example shows how to use GenerateDataKey.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [GenerateDataKey](#) in *AWS SDK for Rust API reference*.

GenerateDataKeyWithoutPlaintext

The following code example shows how to use GenerateDataKeyWithoutPlaintext.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [GenerateDataKeyWithoutPlaintext](#) in *AWS SDK for Rust API reference*.

GenerateRandom

The following code example shows how to use GenerateRandom.

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [GenerateRandom](#) in *AWS SDK for Rust API reference*.

ListKeys

The following code example shows how to use ListKeys.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [ListKeys](#) in *AWS SDK for Rust API reference*.

ReEncrypt

The following code example shows how to use ReEncrypt.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see ReEncrypt in *AWS SDK for Rust API reference.*

Lambda examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Lambda.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Scenarios
- Serverless examples

Actions

CreateFunction

The following code example shows how to use `CreateFunction`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/**
 * Create a function, uploading from a zip file.
 */
```

```rust
    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())
    }
```

- For API details, see [CreateFunction](#) in *AWS SDK for Rust API reference.*

DeleteFunction

The following code example shows how to use `DeleteFunction`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    /** 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)
    }
```

- For API details, see DeleteFunction in *AWS SDK for Rust API reference.*

GetFunction

The following code example shows how to use `GetFunction`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/** 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)
}
```

- For API details, see GetFunction in *AWS SDK for Rust API reference.*

Invoke

The following code example shows how to use `Invoke`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /** 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")
    }
}
```

- For API details, see Invoke in *AWS SDK for Rust API reference*.

ListFunctions

The following code example shows how to use ListFunctions.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
    /** 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)
    }
```

- For API details, see [ListFunctions](#) in *AWS SDK for Rust API reference.*

UpdateFunctionCode

The following code example shows how to use `UpdateFunctionCode`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
    /** 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,
```

```rust
    ) -> 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())
    }
```

- For API details, see UpdateFunctionCode in *AWS SDK for Rust API reference*.

UpdateFunctionConfiguration

The following code example shows how to use UpdateFunctionConfiguration.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

```
/** 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)
```

```
    }
```

- For API details, see UpdateFunctionConfiguration in *AWS SDK for Rust API reference*.

Scenarios

Create a serverless application to manage photos

The following code example shows how to create a serverless application that lets users manage photos using labels.

SDK for Rust

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

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

Get started with functions

The following code example shows how to:

- Create an IAM role and Lambda function, then upload handler code.
- Invoke the function with a single parameter and get results.

- Update the function code and configure with an environment variable.

- Invoke the function with new parameters and get results. Display the returned execution log.

- List the functions for your account, then clean up resources.

For more information, see [Create a Lambda function with the console](#).

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

The Cargo.toml with dependencies used in this scenario.

```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"
```

A collection of utilities that streamline calling Lambda for this scenario. This file is src/ations.rs in the crate.

```rust
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,
}
```

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

```rust
                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);
```

```rust
pub struct OwnBucket(pub bool);

impl LambdaManager {
    pub fn new(
        iam_client: aws_sdk_iam::Client,
        lambda_client: aws_sdk_lambda::Client,
        s3_client: aws_sdk_s3::Client,
        lambda_name: LambdaName,
        role_name: RoleName,
        bucket: Bucket,
        own_bucket: OwnBucket,
    ) -> Self {
        Self {
            iam_client,
            lambda_client,
            s3_client,
            lambda_name: lambda_name.0,
            role_name: role_name.0,
            bucket: bucket.0,
            own_bucket: own_bucket.0,
        }
    }

    /**
     * Load the AWS configuration from the environment.
     * Look up lambda_name and bucket if none are given, or generate a random name
  if not present in the environment.
     * If the bucket name is provided, the caller needs to have created the bucket.
     * If the bucket name is generated, it will be created.
     */
    pub async fn load_from_env(lambda_name: Option<String>, bucket: Option<String>)
 -> Self {
        let sdk_config = aws_config::load_from_env().await;
        let lambda_name = LambdaName(lambda_name.unwrap_or_else(|| {
            std::env::var("LAMBDA_NAME").unwrap_or_else(|_|
 "rust_lambda_example".to_string())
        }));
        let role_name = RoleName(format!("{}_role", lambda_name.0));
        let (bucket, own_bucket) =
            match bucket {
                Some(bucket) => (Bucket(bucket), false),
                None => (
                    Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                        format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
```

```
                    })),
                    true,
                ),
            };

        let s3_client = aws_sdk_s3::Client::new(&sdk_config);

        if own_bucket {
            info!("Creating bucket for demo: {}", bucket.0);
            s3_client
                .create_bucket()
                .bucket(bucket.0.clone())
                .create_bucket_configuration(
                    CreateBucketConfiguration::builder()

 .location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                            sdk_config.region().unwrap().as_ref(),
                        ))
                        .build(),
                )
                .send()
                .await
                .unwrap();
        }

        Self::new(
            aws_sdk_iam::Client::new(&sdk_config),
            aws_sdk_lambda::Client::new(&sdk_config),
            s3_client,
            lambda_name,
            role_name,
            bucket,
            OwnBucket(own_bucket),
        )
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --output-
format Zip`.
     */
    async fn prepare_function(
        &self,
```

```
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }

    /**
     * Create a function, uploading from a zip file.
     */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
            .function_name(self.lambda_name.clone())
            .code(code)
            .role(role.arn())
```

```
            .runtime(aws_sdk_lambda::types::Runtime::Providedal2)
            .handler("_unused")
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        self.lambda_client
            .publish_version()
            .function_name(self.lambda_name.clone())
            .send()
            .await?;

        Ok(key)
    }

    /**
     * Create an IAM execution role for the managed Lambda function.
     * If the role already exists, use that instead.
     */
    async fn create_role(&self) -> Result<aws_sdk_iam::types::Role, CreateRoleError>
{
        info!("Creating execution role for function");
        let get_role = self
            .iam_client
            .get_role()
            .role_name(self.role_name.clone())
            .send()
            .await;
        if let Ok(get_role) = get_role {
            if let Some(role) = get_role.role {
                return Ok(role);
            }
        }

        let create_role = self
            .iam_client
            .create_role()
            .role_name(self.role_name.clone())
            .assume_role_policy_document(ROLE_POLICY_DOCUMENT)
            .send()
            .await;
```

```
        match create_role {
            Ok(create_role) => match create_role.role {
                Some(role) => Ok(role),
                None => Err(CreateRoleError::generic(
                    ErrorMetadata::builder()
                        .message("CreateRole returned empty success")
                        .build(),
                )),
            },
            Err(err) => Err(err.into_service_error()),
        }
    }

    /**
     * Poll `is_function_ready` with a 1-second delay. It returns when the function
    is ready or when there's an error checking the function's state.
     */
    pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
        info!("Waiting for function");
        while !self.is_function_ready(None).await? {
            info!("Function is not ready, sleeping 1s");
            tokio::time::sleep(Duration::from_secs(1)).await;
        }
        Ok(())
    }

    /**
     * Check if a Lambda function is ready to be invoked.
     * A Lambda function is ready for this scenario when its state is active and its
    LastUpdateStatus is Successful.
     * Additionally, if a sha256 is provided, the function must have that as its
    current code hash.
     * Any missing properties or failed requests will be reported as an Err.
     */
    async fn is_function_ready(
        &self,
        expected_code_sha256: Option<&str>,
    ) -> Result<bool, anyhow::Error> {
        match self.get_function().await {
            Ok(func) => {
                if let Some(config) = func.configuration() {
                    if let Some(state) = config.state() {
                        info!(?state, "Checking if function is active");
                        if !matches!(state, State::Active) {
```

```
                                    return Ok(false);
                                }
                            }
                        match config.last_update_status() {
                            Some(last_update_status) => {
                                info!(?last_update_status, "Checking if function is
ready");
                                match last_update_status {
                                    LastUpdateStatus::Successful => {
                                        // continue
                                    }
                                    LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                                        return Ok(false);
                                    }
                                    unknown => {
                                        warn!(
                                            status_variant = unknown.as_str(),
                                            "LastUpdateStatus unknown"
                                        );
                                        return Err(anyhow!(
                                            "Unknown LastUpdateStatus, fn config is
{config:?}"
                                        ));
                                    }
                                }
                            }
                            None => {
                                warn!("Missing last update status");
                                return Ok(false);
                            }
                        };
                        if expected_code_sha256.is_none() {
                            return Ok(true);
                        }
                        if let Some(code_sha256) = config.code_sha256() {
                            return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
                        }
                    }
                }
            Err(e) => {
                warn!(?e, "Could not get function while waiting");
            }
```

```
        }
        Ok(false)
    }

    /** Get the Lambda function with this Manager's name. */
    pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error> {
        info!("Getting lambda function");
        self.lambda_client
            .get_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** List all Lambda functions in the current Region. */
    pub async fn list_functions(&self) -> Result<ListFunctionsOutput, anyhow::Error>
{
        info!("Listing lambda functions");
        self.lambda_client
            .list_functions()
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** Invoke the lambda function using calculator InvokeArgs. */
    pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
        info!(?args, "Invoking {}", self.lambda_name);
        let payload = serde_json::to_string(&args)?;
        debug!(?payload, "Sending payload");
        self.lambda_client
            .invoke()
            .function_name(self.lambda_name.clone())
            .payload(Blob::new(payload))
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
    pub async fn update_function_code(
```

```rust
        &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(),
        );
    }
}
```

A binary to run the scenario from front to end, using command line flags to control some behavior. This file is src/bin/scenario.rs in the crate.

```
/*
## 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.
 */

```rust
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.
```

```rust
    #[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(
```

```rust
            &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");
    }
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.

  - [CreateFunction](#)

  - [DeleteFunction](#)

  - [GetFunction](#)

  - [Invoke](#)

  - [ListFunctions](#)

  - [UpdateFunctionCode](#)

  - [UpdateFunctionConfiguration](#)

Serverless examples

Connecting to an Amazon RDS database in a Lambda function

The following code example shows how to implement a Lambda function that connects to an RDS database. The function makes a simple database request and returns the result.

SDK for Rust

> ℹ️ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Connecting to an Amazon RDS database in a Lambda function using Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
```

```rust
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
    );
```

```rust
    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}")
    }))
}
```

Invoke a Lambda function from a Kinesis trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a Kinesis stream. The function retrieves the Kinesis payload, decodes from Base64, and logs the record contents.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming an Kinesis event with Lambda using Rust.

```
// 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> {
```

```rust
    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
}
```

Invoke a Lambda function from a DynamoDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DynamoDB stream. The function retrieves the DynamoDB payload and logs the record contents.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming a DynamoDB event with Lambda using Rust.

```rust
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(())

    }
```

Invoke a Lambda function from a Amazon DocumentDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DocumentDB change stream. The function retrieves the DocumentDB payload and logs the record contents.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [Serverless examples](#) repository.

Consuming a Amazon DocumentDB event with Lambda using Rust.

```rust
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(())

}
```

Invoke a Lambda function from an Amazon S3 trigger

The following code example shows how to implement a Lambda function that receives an event triggered by uploading an object to an S3 bucket. The function retrieves the S3 bucket name and object key from the event parameter and calls the Amazon S3 API to retrieve and log the content type of the object.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an S3 event with Lambda using Rust.

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

Invoke a Lambda function from an Amazon SNS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SNS topic. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming an SNS event with Lambda using Rust.

```
// 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;
```

```rust
// Built with the following dependencies:
//  aws_lambda_events = { version = "0.10.0", default-features = false, features =
  ["sns"] }
//  lambda_runtime = "0.8.1"
//  tokio = { version = "1", features = ["macros"] }
//  tracing = { version = "0.1", features = ["log"] }
//  tracing-subscriber = { version = "0.3", default-features = false, features =
  ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);

    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Invoke a Lambda function from an Amazon SQS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SQS queue. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming an SQS event with Lambda using Rust.

```rust
// 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
}
```

Reporting batch item failures for Lambda functions with a Kinesis trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a Kinesis stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Reporting Kinesis batch item failures with Lambda using Rust.

```rust
// 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()
        );
```

```rust
        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
}
```

Reporting batch item failures for Lambda functions with a DynamoDB trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a DynamoDB stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Reporting DynamoDB batch item failures with Lambda using Rust.

```
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...
```

```rust
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
}
```

Reporting batch item failures for Lambda functions with an Amazon SQS trigger

The following code example shows how to implement partial batch response for Lambda functions
that receive events from an SQS queue. The function reports the batch item failures in the
response, signaling to Lambda to retry those messages later.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the Serverless examples repository.

Reporting SQS batch item failures with Lambda using Rust.

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

```rust
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
}
```

MediaLive examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with MediaLive.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

`ListInputs`

The following code example shows how to use `ListInputs`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List your MediaLive input names and ARNs in the Region.

```
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(())
}
```

- For API details, see [ListInputs](#) in *AWS SDK for Rust API reference*.

MediaPackage examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with MediaPackage.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

`ListChannels`

The following code example shows how to use `ListChannels`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List channel ARNs and descriptions.

```
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(())
}
```

- For API details, see [ListChannels](#) in *AWS SDK for Rust API reference*.

ListOriginEndpoints

The following code example shows how to use `ListOriginEndpoints`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List your endpoint descriptions and URLs.

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

- For API details, see [ListOriginEndpoints](#) in *AWS SDK for Rust API reference.*

Amazon Polly examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon Polly.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Scenarios

Actions

DescribeVoices

The following code example shows how to use `DescribeVoices`.

SDK for Rust

> ℹ️ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see [DescribeVoices](#) in *AWS SDK for Rust API reference.*

ListLexicons

The following code example shows how to use `ListLexicons`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [ListLexicons](#) in *AWS SDK for Rust API reference*.

PutLexicon

The following code example shows how to use `PutLexicon`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
  }
```

- For API details, see [PutLexicon](#) in *AWS SDK for Rust API reference*.

SynthesizeSpeech

The following code example shows how to use SynthesizeSpeech.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [SynthesizeSpeech](#) in *AWS SDK for Rust API reference*.

Scenarios

Convert text to speech and back to text

The following code example shows how to:

- Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file.

- Upload the audio file to an Amazon S3 bucket.

- Use Amazon Transcribe to convert the audio file to text.

- Display the text.

SDK for Rust

Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file, upload the audio file to an Amazon S3 bucket, use Amazon Transcribe to convert that audio file to text, and display the text.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Polly

- Amazon S3

- Amazon Transcribe

QLDB examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with QLDB.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

`CreateLedger`

The following code example shows how to use `CreateLedger`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see CreateLedger in *AWS SDK for Rust API reference.*

ListLedgers

The following code example shows how to use `ListLedgers`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see ListLedgers in *AWS SDK for Rust API reference.*

Amazon RDS examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon RDS.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Serverless examples](#)

Serverless examples

Connecting to an Amazon RDS database in a Lambda function

The following code example shows how to implement a Lambda function that connects to an RDS database. The function makes a simple database request and returns the result.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Connecting to an Amazon RDS database in a Lambda function using Rust.

```
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());
```

```rust
        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}")
```

```
        }))
}
```

Amazon RDS Data Service examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon RDS Data Service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

ExecuteStatement

The following code example shows how to use `ExecuteStatement`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see ExecuteStatement in *AWS SDK for Rust API reference*.

Amazon Rekognition examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon Rekognition.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Scenarios

Scenarios

Create a serverless application to manage photos

The following code example shows how to create a serverless application that lets users manage photos using labels.

SDK for Rust

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

For a deep dive into the origin of this example see the post on [AWS Community](#).

Services used in this example

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

Detect faces in an image

The following code example shows how to:

- Save an image in an Amazon S3 bucket.
- Use Amazon Rekognition to detect facial details, such as age range, gender, and emotion (such as smiling).
- Display those details.

SDK for Rust

Save the image in an Amazon S3 bucket with an **uploads** prefix, use Amazon Rekognition to detect facial details, such as age range, gender, and emotion (smiling, etc.), and display those details.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition

- Amazon S3

Save EXIF and other image information

The following code example shows how to:

- Get EXIF information from a a JPG, JPEG, or PNG file.
- Upload the image file to an Amazon S3 bucket.
- Use Amazon Rekognition to identify the three top attributes (labels) in the file.
- Add the EXIF and label information to an Amazon DynamoDB table in the Region.

SDK for Rust

Get EXIF information from a JPG, JPEG, or PNG file, upload the image file to an Amazon S3 bucket, use Amazon Rekognition to identify the three top attributes (*labels* in Amazon Rekognition) in the file, and add the EXIF and label information to a Amazon DynamoDB table in the Region.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- DynamoDB
- Amazon Rekognition
- Amazon S3

Route 53 examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Route 53.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

ListHostedZones

The following code example shows how to use `ListHostedZones`.

SDK for Rust

> ℹ️ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [ListHostedZones](#) in *AWS SDK for Rust API reference*.

Amazon S3 examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon S3.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Amazon S3

The following code examples show how to get started using Amazon S3.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// 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 utilities

```
/// S3ExampleError provides a From<T: ProvideErrorMetadata> impl to extract
/// client-specific error details. This serves as a consistent backup to handling
/// specific service errors, depending on what is needed by the scenario.
/// It is used throughout the code examples for the AWS SDK for Rust.
#[derive(Debug)]
pub struct S3ExampleError(String);
impl S3ExampleError {
    pub fn new(value: impl Into<String>) -> Self {
        S3ExampleError(value.into())
    }

    pub fn add_message(self, message: impl Into<String>) -> Self {
        S3ExampleError(format!("{}: {}", message.into(), self.0))
    }
}

impl<T: aws_sdk_s3::error::ProvideErrorMetadata> From<T> for S3ExampleError {
    fn from(value: T) -> Self {
        S3ExampleError(format!(
```

```
                "{}: {}",
                value
                    .code()
                    .map(String::from)
                    .unwrap_or("unknown code".into()),
                value
                    .message()
                    .map(String::from)
                    .unwrap_or("missing reason".into()),
        ))
    }
}

impl std::error::Error for S3ExampleError {}

impl std::fmt::Display for S3ExampleError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
```

- For API details, see [ListBuckets](#) in *AWS SDK for Rust API reference.*

Topics

- [Basics](#)
- [Actions](#)
- [Scenarios](#)
- [Serverless examples](#)

Basics

Learn the basics

The following code example shows how to:

- Create a bucket and upload a file to it.
- Download an object from a bucket.
- Copy an object to a subfolder in a bucket.

- List the objects in a bucket.

- Delete the bucket objects and the bucket.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Code for the binary crate which runs the scenario.

```rust
#![allow(clippy::result_large_err)]

//!  Purpose
//!  Shows how to use the AWS SDK for Rust to get started using
//!  Amazon Simple Storage Service (Amazon S3). Create a bucket, move objects into
//!  and out of it,
//!  and delete all resources at the end of the demo.
//!
//!  This example follows the steps in "Getting started with Amazon S3" in the
//!  Amazon S3
//!  user guide.
//!  - https://docs.aws.amazon.com/AmazonS3/latest/userguide/GetStartedWithS3.html

use aws_config::meta::region::RegionProviderChain;
use aws_sdk_s3::{config::Region, Client};
use s3_code_examples::error::S3ExampleError;
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), S3ExampleError> {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
    let region = region_provider.region().await.unwrap();
    let shared_config = aws_config::from_env().region(region_provider).load().await;
    let client = Client::new(&shared_config);
    let bucket_name = format!("amzn-s3-demo-bucket-{}", Uuid::new_v4());
    let file_name = "s3/testfile.txt".to_string();
    let key = "test file key name".to_string();
```

```rust
    let target_key = "target_key".to_string();

    if let Err(e) = run_s3_operations(region, client, bucket_name, file_name, key,
 target_key).await
    {
        eprintln!("{:?}", e);
    };

    Ok(())
}

async fn run_s3_operations(
    region: Region,
    client: Client,
    bucket_name: String,
    file_name: String,
    key: String,
    target_key: String,
) -> Result<(), S3ExampleError> {
    s3_code_examples::create_bucket(&client, &bucket_name, &region).await?;
    let run_example: Result<(), S3ExampleError> = (async {
        s3_code_examples::upload_object(&client, &bucket_name, &file_name,
 &key).await?;
        let _object = s3_code_examples::download_object(&client, &bucket_name,
 &key).await;
        s3_code_examples::copy_object(&client, &bucket_name, &bucket_name, &key,
 &target_key)
            .await?;
        s3_code_examples::list_objects(&client, &bucket_name).await?;
        s3_code_examples::clear_bucket(&client, &bucket_name).await?;
        Ok(())
    })
    .await;
    if let Err(err) = run_example {
        eprintln!("Failed to complete getting-started example: {err:?}");
    }
    s3_code_examples::delete_bucket(&client, &bucket_name).await?;

    Ok(())
}
```

Common actions used by the scenario.

```rust
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
```

```rust
        .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))
            }
        }
```

```
        }
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.

  - CopyObject

  - CreateBucket

  - DeleteBucket

  - DeleteObjects

  - GetObject

  - ListObjectsV2

  - PutObject

Actions

CompleteMultipartUpload

The following code example shows how to use `CompleteMultipartUpload`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see CompleteMultipartUpload in *AWS SDK for Rust API reference*.

CopyObject

The following code example shows how to use CopyObject.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

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

- For API details, see [CopyObject](#) in *AWS SDK for Rust API reference*.

CreateBucket

The following code example shows how to use CreateBucket.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
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))
        }
    })
}
```

- For API details, see [CreateBucket](#) in *AWS SDK for Rust API reference.*

CreateMultipartUpload

The following code example shows how to use `CreateMultipartUpload`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
    // 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?;
```

- For API details, see CreateMultipartUpload in *AWS SDK for Rust API reference*.

DeleteBucket

The following code example shows how to use DeleteBucket.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

```
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))
            }
        }
    }
}
```

- For API details, see [DeleteBucket](#) in *AWS SDK for Rust API reference.*

DeleteObject

The following code example shows how to use `DeleteObject`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see DeleteObject in *AWS SDK for Rust API reference.*

DeleteObjects

The following code example shows how to use DeleteObjects.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

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

- For API details, see [DeleteObjects](#) in *AWS SDK for Rust API reference*.

GetBucketLocation

The following code example shows how to use GetBucketLocation.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [GetBucketLocation](#) in *AWS SDK for Rust API reference*.

GetObject

The following code example shows how to use GetObject.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](.).

```rust
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)
```

```
}
```

- For API details, see GetObject in *AWS SDK for Rust API reference.*

ListBuckets

The following code example shows how to use `ListBuckets`.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see [ListBuckets](#) in *AWS SDK for Rust API reference*.

ListObjectVersions

The following code example shows how to use `ListObjectVersions`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
    }
```

- For API details, see [ListObjectVersions](#) in *AWS SDK for Rust API reference*.

ListObjectsV2

The following code example shows how to use `ListObjectsV2`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see ListObjectsV2 in *AWS SDK for Rust API reference*.

PutObject

The following code example shows how to use `PutObject`.

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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)
}
```

- For API details, see PutObject in *AWS SDK for Rust API reference*.

UploadPart

The following code example shows how to use `UploadPart`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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?;
```

- For API details, see [UploadPart](#) in *AWS SDK for Rust API reference.*

Scenarios

Convert text to speech and back to text

The following code example shows how to:

- Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file.

- Upload the audio file to an Amazon S3 bucket.

- Use Amazon Transcribe to convert the audio file to text.

- Display the text.

SDK for Rust

Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file, upload the audio file to an Amazon S3 bucket, use Amazon Transcribe to convert that audio file to text, and display the text.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Polly

- Amazon S3

- Amazon Transcribe

Create a presigned URL

The following code example shows how to create a presigned URL for Amazon S3 and upload an object.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create presigning requests to GET S3 objects.

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

Create presigning requests to PUT S3 objects.

```
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())
}
```

Create a serverless application to manage photos

The following code example shows how to create a serverless application that lets users manage photos using labels.

SDK for Rust

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

For a deep dive into the origin of this example see the post on [AWS Community](#).

Services used in this example

- API Gateway

- DynamoDB

- Lambda

- Amazon Rekognition

- Amazon S3

- Amazon SNS

Detect faces in an image

The following code example shows how to:

- Save an image in an Amazon S3 bucket.

- Use Amazon Rekognition to detect facial details, such as age range, gender, and emotion (such as smiling).

- Display those details.

SDK for Rust

Save the image in an Amazon S3 bucket with an **uploads** prefix, use Amazon Rekognition to detect facial details, such as age range, gender, and emotion (smiling, etc.), and display those details.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3

Get an object from a bucket if it has been modified

The following code example shows how to read data from an object in an S3 bucket, but only if that bucket has not been modified since the last retrieval time.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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
```

```rust
/// - 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()
        }
    };
```

```rust
    // Request the object's headers.
    let head_object_output = client
        .head_object()
        .bucket(bucket_name.as_str())
        .key(KEY)
        .send()
        .await;

    // If the `HeadObject` request succeeded, create a tuple containing the
    // values of the headers `last-modified` and `etag`. If the request
    // failed, return the error in a tuple instead.
    let (last_modified, e_tag_2) = match head_object_output {
        Ok(head_object) => (
            Ok(head_object.last_modified().cloned().unwrap()),
            head_object.e_tag.unwrap(),
        ),
        Err(err) => (Err(err), String::new()),
    };

    warn!("last modified: {last_modified:?}");
    assert_eq!(
        e_tag_1, e_tag_2,
        "PutObject and first GetObject had differing eTags"
    );

    println!("First value of last_modified: {last_modified:?}");
    println!("First tag: {}\n", e_tag_1);

    // Send a second `HeadObject` request. This time, the `if_modified_since`
    // option is specified, giving the `last_modified` value returned by the
    // first call to `HeadObject`.
    //
    // Since the object hasn't been changed, and there are no other objects in
    // the bucket, there should be no matching objects.

    let head_object_output = client
        .head_object()
        .bucket(bucket_name.as_str())
        .key(KEY)
        .if_modified_since(last_modified.unwrap())
        .send()
        .await;

    // If the `HeadObject` request succeeded, the result is a typle containing
```

```
        // the `last_modified` and `e_tag_1` properties. This is _not_ the expected
        // result.
        //
        // The _expected_ result of the second call to `HeadObject` is an
        // `SdkError::ServiceError` containing the HTTP error response. If that's
        // the case and the HTTP status is 304 (not modified), the output is a
        // tuple containing the values of the HTTP `last-modified` and `etag`
        // headers.
        //
        // If any other HTTP error occurred, the error is returned as an
        // `SdkError::ServiceError`.

        let (last_modified, e_tag_2) = match head_object_output {
            Ok(head_object) => (
                Ok(head_object.last_modified().cloned().unwrap()),
                head_object.e_tag.unwrap(),
            ),
            Err(err) => match err {
                SdkError::ServiceError(err) => {
                    // Get the raw HTTP response. If its status is 304, the
                    // object has not changed. This is the expected code path.
                    let http = err.raw();
                    match http.status().as_u16() {
                        // If the HTTP status is 304: Not Modified, return a
                        // tuple containing the values of the HTTP
                        // `last-modified` and `etag` headers.
                        304 => (
                            Ok(DateTime::from_str(
                                http.headers().get("last-modified").unwrap(),
                                DateTimeFormat::HttpDate,
                            )
                            .unwrap()),
                            http.headers().get("etag").map(|t| t.into()).unwrap(),
                        ),
                        // Any other HTTP status code is returned as an
                        // `SdkError::ServiceError`.
                        _ => (Err(SdkError::ServiceError(err)), String::new()),
                    }
                }
                // Any other kind of error is returned in a tuple containing the
                // error and an empty string.
                _ => (Err(err), String::new()),
            },
        };
```

```
    warn!("last modified: {last_modified:?}");
    assert_eq!(
        e_tag_1, e_tag_2,
        "PutObject and second HeadObject had different eTags"
    );

    println!("Second value of last modified: {last_modified:?}");
    println!("Second tag: {}", e_tag_2);

    // Clean up by deleting the object and the bucket.
    client
        .delete_object()
        .bucket(bucket_name.as_str())
        .key(KEY)
        .send()
        .await?;

    client
        .delete_bucket()
        .bucket(bucket_name.as_str())
        .send()
        .await?;

    Ok(())
}
```

- For API details, see GetObject in *AWS SDK for Rust API reference.*

Save EXIF and other image information

The following code example shows how to:

- Get EXIF information from a a JPG, JPEG, or PNG file.

- Upload the image file to an Amazon S3 bucket.

- Use Amazon Rekognition to identify the three top attributes (labels) in the file.

- Add the EXIF and label information to an Amazon DynamoDB table in the Region.

SDK for Rust

Get EXIF information from a JPG, JPEG, or PNG file, upload the image file to an Amazon
S3 bucket, use Amazon Rekognition to identify the three top attributes (*labels* in Amazon
Rekognition) in the file, and add the EXIF and label information to a Amazon DynamoDB table
in the Region.

For complete source code and instructions on how to set up and run, see the full example on
[GitHub](#).

Services used in this example

- DynamoDB

- Amazon Rekognition

- Amazon S3

Unit and integration test with an SDK

The following code example shows how to examples for best-practice techniques when writing unit
and integration tests using an AWS SDK.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

Cargo.toml for testing examples.

```toml
[package]
name = "testing-examples"
version = "0.1.0"
authors = [
  "John Disanti <jdisanti@amazon.com>",
  "Doug Schwartz <dougsch@amazon.com>",
]
edition = "2021"
```

```
[dependencies]
async-trait = "0.1.51"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-credential-types = { version = "1.0.1", features = [ "hardcoded-
credentials", ] }
aws-sdk-s3 = { version = "1.4.0" }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime = { version = "1.0.1", features = ["test-util"] }
aws-smithy-runtime-api = { version = "1.0.1", features = ["test-util"] }
aws-types = { version = "1.0.1" }
clap = { version = "~4.4", features = ["derive"] }
http = "0.2.9"
mockall = "0.11.4"
serde_json = "1"
tokio = { version = "1.20.1", features = ["full"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }

[[bin]]
name = "main"
path = "src/main.rs"
```

Unit testing example using automock and a service wrapper.

```
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 {
```

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

Integration testing example using StaticReplayClient.

```
use aws_sdk_s3 as s3;

#[allow(dead_code)]
pub async fn determine_prefix_file_size(
    // Now we take a reference to our trait object instead of the S3 client
    // s3_list: ListObjectsService,
```

```
        s3: s3::Client,
        bucket: &str,
        prefix: &str,
    ) -> Result<usize, s3::Error> {
        let mut next_token: Option<String> = None;
        let mut total_size_bytes = 0;
        loop {
            let result = s3
                .list_objects_v2()
                .prefix(prefix)
                .bucket(bucket)
                .set_continuation_token(next_token.take())
                .send()
                .await?;

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

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

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

#[cfg(test)]
mod test {
    use super::*;
    use aws_config::BehaviorVersion;
```

```rust
    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")
```

```rust
                    .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(&[]);
    }
}
```

Upload or download large files

The following code example shows how to upload or download large files to and from Amazon S3.

For more information, see [Uploading an object using multipart upload](#).

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```rust
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() {
```

```rust
    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)
```

```rust
            .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(())
}
```

Serverless examples

Invoke a Lambda function from an Amazon S3 trigger

The following code example shows how to implement a Lambda function that receives an event triggered by uploading an object to an S3 bucket. The function retrieves the S3 bucket name and object key from the event parameter and calls the Amazon S3 API to retrieve and log the content type of the object.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming an S3 event with Lambda using Rust.

```rust
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};


/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
```

```
            .with_target(false)
            .without_time()
            .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request from
 SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket = evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket name
 to exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object key to
 exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
 contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
```

```
    }

    Ok(())
}
```

SageMaker examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with SageMaker.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

ListNotebookInstances

The following code example shows how to use `ListNotebookInstances`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [ListNotebookInstances](#) in *AWS SDK for Rust API reference.*

ListTrainingJobs

The following code example shows how to use `ListTrainingJobs`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [ListTrainingJobs](#) in *AWS SDK for Rust API reference.*

Secrets Manager examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Secrets Manager.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

GetSecretValue

The following code example shows how to use GetSecretValue.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [GetSecretValue](#) in *AWS SDK for Rust API reference*.

Amazon SES API v2 examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon SES API v2.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)
- [Scenarios](#)

Actions

CreateContact

The following code example shows how to use `CreateContact`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see CreateContact in *AWS SDK for Rust API reference*.

CreateContactList

The following code example shows how to use `CreateContactList`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see CreateContactList in *AWS SDK for Rust API reference*.

CreateEmailIdentity

The following code example shows how to use `CreateEmailIdentity`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

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

- For API details, see [CreateEmailIdentity](#) in *AWS SDK for Rust API reference*.

CreateEmailTemplate

The following code example shows how to use CreateEmailTemplate.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [CreateEmailTemplate](#) in *AWS SDK for Rust API reference*.

DeleteContactList

The following code example shows how to use DeleteContactList.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
        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}")),
        }
```

- For API details, see DeleteContactList in *AWS SDK for Rust API reference*.

DeleteEmailIdentity

The following code example shows how to use DeleteEmailIdentity.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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));
            }
        }
```

- For API details, see DeleteEmailIdentity in *AWS SDK for Rust API reference*.

DeleteEmailTemplate

The following code example shows how to use DeleteEmailTemplate.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see [DeleteEmailTemplate](#) in *AWS SDK for Rust API reference*.

GetEmailIdentity

The following code example shows how to use `GetEmailIdentity`.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Determines whether an email address has been verified.

```
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(())
}
```

- For API details, see GetEmailIdentity in *AWS SDK for Rust API reference*.

ListContactLists

The following code example shows how to use `ListContactLists`.

SDK for Rust

> (i) **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see ListContactLists in *AWS SDK for Rust API reference*.

ListContacts

The following code example shows how to use `ListContacts`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

- For API details, see ListContacts in *AWS SDK for Rust API reference*.

SendEmail

The following code example shows how to use `SendEmail`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Sends a message to all members of the contact list.

```rust
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");
```

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

Sends a message to all members of the contact list using a template.

```rust
            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)),
        }
```

- For API details, see [SendEmail](#) in *AWS SDK for Rust API reference.*

Scenarios

Newsletter workflow

The following code example shows how to run the Amazon SES API v2 newsletter workflow.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

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

- For API details, see the following topics in *AWS SDK for Rust API reference*.

  - [CreateContact](#)

  - [CreateContactList](#)

  - [CreateEmailIdentity](#)

  - [CreateEmailTemplate](#)

  - [DeleteContactList](#)

  - [DeleteEmailIdentity](#)

  - [DeleteEmailTemplate](#)

  - [ListContacts](#)

  - [SendEmail.simple](#)

  - [SendEmail.template](#)

Amazon SNS examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon SNS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Scenarios
- Serverless examples

Actions

CreateTopic

The following code example shows how to use `CreateTopic`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see CreateTopic in *AWS SDK for Rust API reference*.

ListTopics

The following code example shows how to use `ListTopics`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see [ListTopics](#) in *AWS SDK for Rust API reference.*

Publish

The following code example shows how to use `Publish`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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(())
}
```

- For API details, see Publish in *AWS SDK for Rust API reference.*

Subscribe

The following code example shows how to use Subscribe.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
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(())
}
```

- For API details, see Subscribe in *AWS SDK for Rust API reference*.

Scenarios

Create a serverless application to manage photos

The following code example shows how to create a serverless application that lets users manage photos using labels.

SDK for Rust

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

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

Serverless examples

Invoke a Lambda function from an Amazon SNS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SNS topic. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
```

```rust
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
//   aws_lambda_events = { version = "0.10.0", default-features = false, features =
//  ["sns"] }
//   lambda_runtime = "0.8.1"
//   tokio = { version = "1", features = ["macros"] }
//   tracing = { version = "0.1", features = ["log"] }
//   tracing-subscriber = { version = "0.3", default-features = false, features =
//  ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);

    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Amazon SQS examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon SQS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Serverless examples

Actions

ListQueues

The following code example shows how to use `ListQueues`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Retrieve the first Amazon SQS queue listed in the Region.

```
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())
}
```

- For API details, see ListQueues in *AWS SDK for Rust API reference*.

ReceiveMessage

The following code example shows how to use `ReceiveMessage`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see ReceiveMessage in *AWS SDK for Rust API reference*.

SendMessage

The following code example shows how to use `SendMessage`.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see SendMessage in *AWS SDK for Rust API reference*.

Serverless examples

Invoke a Lambda function from an Amazon SQS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SQS queue. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using Rust.

```
// 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
}
```

Reporting batch item failures for Lambda functions with an Amazon SQS trigger

The following code example shows how to implement partial batch response for Lambda functions
that receive events from an SQS queue. The function reports the batch item failures in the
response, signaling to Lambda to retry those messages later.

SDK for Rust

> ### ⓘ Note
>
> There's more on GitHub. Find the complete example and learn how to set up and run in
> the [Serverless examples](#) repository.

Reporting SQS batch item failures with Lambda using Rust.

```rust
// 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 examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with AWS STS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

AssumeRole

The following code example shows how to use `AssumeRole`.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```rust
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),
    }
}
```

- For API details, see AssumeRole in *AWS SDK for Rust API reference*.

Systems Manager examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Systems Manager.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

DescribeParameters

The following code example shows how to use DescribeParameters.

SDK for Rust

> ⓘ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see [DescribeParameters](#) in *AWS SDK for Rust API reference*.

GetParameter

The following code example shows how to use GetParameter.

SDK for Rust

> **ⓘ Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    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)
    }
```

- For API details, see GetParameter in *AWS SDK for Rust API reference*.

PutParameter

The following code example shows how to use `PutParameter`.

SDK for Rust

> ℹ **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see PutParameter in *AWS SDK for Rust API reference*.

Amazon Transcribe examples using SDK for Rust

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Rust with Amazon Transcribe.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- [Scenarios](#)

Scenarios

Convert text to speech and back to text

The following code example shows how to:

- Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file.
- Upload the audio file to an Amazon S3 bucket.
- Use Amazon Transcribe to convert the audio file to text.
- Display the text.

SDK for Rust

> Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file, upload the audio file to an Amazon S3 bucket, use Amazon Transcribe to convert that audio file to text, and display the text.
>
> For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).
>
> **Services used in this example**
>
> - Amazon Polly
> - Amazon S3

- Amazon Transcribe

Security for this AWS Product or Service

Cloud security at Amazon Web Services (AWS) is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. The [Shared Responsibility Model](#) describes this as Security of the Cloud and Security in the Cloud.

Security of the Cloud – AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the [AWS Compliance Programs](#).

Security in the Cloud – Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

Topics

- [Data protection in this AWS Product or Service](#)
- [Compliance Validation for this AWS Product or Service](#)
- [Infrastructure Security for this AWS Product or Service](#)
- [Enforce a minimum TLS version in the AWS SDK for Rust](#)

Data protection in this AWS Product or Service

The AWS [shared responsibility model](#) applies to data protection in this AWS product or service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.

- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.

- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.

- Use AWS encryption solutions, along with all default security controls within AWS services.

- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.

- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard (FIPS) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with this AWS product or service or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Compliance Validation for this AWS Product or Service

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.

- **Architecting for HIPAA Security and Compliance on Amazon Web Services** – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

  > **ⓘ Note**
  >
  > Not all AWS services are HIPAA eligible. For more information, see the **HIPAA Eligible Services Reference**.

- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.

- **AWS Customer Compliance Guides** – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- **Evaluating Resources with Rules** in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.

- **AWS Security Hub** – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see **Security Hub controls reference**.

- **Amazon GuardDuty** – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.

- **AWS Audit Manager** – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

This AWS product or service follows the **shared responsibility model** through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the **AWS service**

Infrastructure Security for this AWS Product or Service

This AWS product or service uses managed services, and therefore is protected by the AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access this AWS Product or Service through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Enforce a minimum TLS version in the AWS SDK for Rust

The AWS SDK for Rust uses TLS to increase security when communicating with AWS services. The SDK enforces a minimum TLS version of 1.2 by default. By default, the SDK also negotiates the highest version of TLS available to both the client application and the service. For example, the SDK might be able to negotiate TLS 1.3.

A particular TLS version can be enforced in the application by providing manual configuration of the TCP connector that the SDK uses. To illustrate this, the following example shows you how to enforce TLS 1.3.

> ⓘ **Note**
>
> Some AWS services do not yet support TLS 1.3, so enforcing this version might affect
> SDK interoperability. We recommend testing this configuration with each service before
> production deployment.

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

Appendix: Rust crates used by the SDK

This appendix contains advanced information about the crates used by the AWS SDK for Rust. This includes the Smithy components it uses, crates you might need to use under certain build circumstances, and other information.

Smithy crates

The AWS SDK for Rust is based on Smithy, like most of the AWS SDKs. Smithy is a language used to describe the data types and functions offered by the SDK. These models are then used to help build the SDK itself.

When looking at versions of the SDK for Rust crates and those of its Smithy dependencies, it might be helpful to know that these crates all use standard semantic version numbering.

For additional detailed information about Smithy crates for Rust, see Smithy Rust Design.

Crates used with the SDK for Rust

There are a number of Smithy crates published by AWS. Some of these are relevant to SDK for Rust users, while others are implementation details:

`aws-smithy-async`

　　Include this crate if you're not using Tokio for asynchronous functionality.

`aws-smithy-runtime`

　　Includes building blocks required by all AWS SDKs.

`aws-smithy-runtime-api`

　　Underlying interfaces used by the SDK.

`aws-smithy-types`

　　Types re-exported from other AWS SDKs. Use this if you use multiple SDKs.

`aws-smithy-types-convert`

　　Utility functions for moving in and out of `aws-smithy-types`.

Other crates

The following crates exist, but you should not need to know anything about them:

Server-related crates that SDK for Rust users don't need:

- `aws-smithy-http-server`
- `aws-smithy-http-server-python`

Crates that contain under-the-hood code that SDK users don't need to use:

- `aws-smithy-checksum-callbacks`
- `aws-smithy-eventstream`
- `aws-smithy-http`
- `aws-smithy-protocol-test`
- `aws-smithy-query`
- `aws-smithy-json`
- `aws-smithy-xml`

Crates that are unsupported and will go away in the future:

- `aws-smithy-client`
- `aws-smithy-http-auth`
- `aws-smithy-http-tower`

Document history

This topic describes important changes to the AWS SDK for Rust Developer Guide over the course of its history.

Change	Description	Date
Reorganizing the Table of Contents	Reorganizing the Table of Contents to better different iate configuration from usage.	July 1, 2024
General availability of the AWS SDK for Rust	Updated the guide to include new security information, new and updated code examples, new details on unit testing with examples, and other new and updated content for the new General Availability release of the SDK.	November 27, 2023
Enforcing a minimum TLS version	Added information about how to enforce a version of TLS in the SDK.	May 4, 2022
AWS SDK for Rust developer preview release	Developer preview release	December 2, 2021